

Document made available under the Patent Cooperation Treaty (PCT)

International application number: PCT/US05/015941

International filing date: 04 May 2005 (04.05.2005)

Document type: Certified copy of priority document

Document details: Country/Office: US
Number: 60/567,980
Filing date: 04 May 2004 (04.05.2004)

Date of receipt at the International Bureau: 16 September 2005 (16.09.2005)

Remark: Priority document submitted or transmitted to the International Bureau in compliance with Rule 17.1(a) or (b)



World Intellectual Property Organization (WIPO) - Geneva, Switzerland
Organisation Mondiale de la Propriété Intellectuelle (OMPI) - Genève, Suisse

1364879

THE UNITED STATES OF AMERICA

TO ALL TO WHOM THESE PRESENTS SHALL COME:

UNITED STATES DEPARTMENT OF COMMERCE

United States Patent and Trademark Office

September 06, 2005

THIS IS TO CERTIFY THAT ANNEXED HERETO IS A TRUE COPY FROM THE RECORDS OF THE UNITED STATES PATENT AND TRADEMARK OFFICE OF THOSE PAPERS OF THE BELOW IDENTIFIED PATENT APPLICATION THAT MET THE REQUIREMENTS TO BE GRANTED A FILING DATE.

APPLICATION NUMBER: 60/567,980

FILING DATE: May 04, 2004

RELATED PCT APPLICATION NUMBER: PCT/US05/15941



Certified by

Under Secretary of Commerce
for Intellectual Property
and Director of the United States
Patent and Trademark Office



PROVISIONAL PATENT APPLICATION
IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Docket No: 20040/59-11538

PROVISIONAL PATENT APPLICATION TRANSMITTAL

Mail Stop Provisional Patent Application
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450



Sir:

Transmitted herewith for filing is the provisional patent application under 37 CFR 1.53(c) of

Inventor: **Mark J. Nixon**

Residence: 1503 Blackjack Drive, Round Rock, Texas 78681

Title: **"Graphical User Interface for Representing, Monitoring,
and Interfacing With Process Control Systems"**

1. Application Papers Enclosed

- 1 Title Page
- 603 Pages of Specification
(excluding Claims, Abstract, Sequence Listing & Drawings)

☐ Formal
☒ Informal

CERTIFICATION UNDER 37 CFR 1.10

I hereby certify that this Provisional Patent Application Transmittal and the documents referred to as enclosed therewith are being deposited with the United States Postal Service on **May 04, 2004** in an envelope addressed to Mail Stop Provisional Patent Application, Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450 utilizing the "Express Mail Post Office to Addressee" service of the United States Postal Service under Mailing Label No. EV 403728624 U.S.


Charissa D. Wheeler

2. Small Entity Status

- ☐ Applicant claims small entity status. See 37 CFR 1.27.
☐ A small entity statement is(are) attached.

3. Additional Papers Enclosed

- ☐ Declaration of Biological Deposit
☐ Computer readable copy of sequence listing containing nucleotide and/or amino acid sequence
☐ Microfiche computer program
☐ Associate power of attorney
☐ Verified translation of a non-English patent application
☐ An assignment of the invention
☒ Return Receipt Postcard
☐ Other (e.g. declaration or oath)

4. Filing Fee Calculation (37 CFR 1.16)

- ☐ Provisional Application (\$80.00/\$160.00) Filing Fee: \$160.00
☐ Other Fees (e.g. Recording Assignment) Filing Fee:

Total Fees Enclosed: **\$160.00**

5. Method of Payment of Fees

- ☒ Enclosed is our firm check in the amount of: \$160.00
☐ Charge Deposit Account No. 50-2455 in the amount of: \$
A copy of this Transmittal is enclosed.

6. Deposit Account and Refund Authorization

The Commissioner is hereby authorized to charge any deficiency in the amount enclosed or any additional fees which may be required during the pendency of this application under 37 CFR 1.16 or 37 CFR 1.17 or under other applicable rules (except payment of issue fees), to Deposit Account No. 50-2455. A copy of this Transmittal is enclosed.

Please refund any overpayment to Grossman & Flight, LLC at the address below.

7. Correspondence Address

Respectfully submitted,

GROSSMAN & FLIGHT, LLC
20 North Wacker Drive
Suite 4220
Chicago, Illinois 60606
(312) 580-1020

By:


Mark G. Hanley
Registration No.: 44,736

May 4, 2004

This Page Is Inserted by IFW Operations
and is not a part of the Official Record

BEST AVAILABLE IMAGES

Defective images within this document are accurate representations of the original documents submitted by the applicant.

Defects in the images may include (but are not limited to):

- BLACK BORDERS
- TEXT CUT OFF AT TOP, BOTTOM OR SIDES
- FADED TEXT
- ILLEGIBLE TEXT
- SKEWED/SLANTED IMAGES
- COLORED PHOTOS
- BLACK OR VERY BLACK AND WHITE DARK PHOTOS
- GRAY SCALE DOCUMENTS

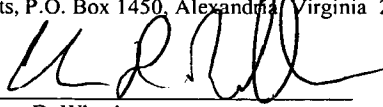
IMAGES ARE BEST AVAILABLE COPY.

**As rescanning documents *will not* correct images,
please do not report the images to the
Image Problem Mailbox.**

"EXPRESS MAIL" mailing label No. EV 403728624 US.

Date of Deposit: May 4, 2004

I hereby certify that this paper (or fee) is being deposited with the United States Postal Service "EXPRESS MAIL POST OFFICE TO ADDRESSEE" service under 37 CFR §1.10 on the date indicated above and is addressed to: Mail Stop Provisional Patent Application, Commissioner for Patents, P.O. Box 1450, Alexandria, Virginia 22313-1450



Charissa D. Wheeler

PROVISIONAL APPLICATION FOR UNITED STATES LETTERS PATENT

SPECIFICATION

TO ALL WHOM IT MAY CONCERN:

Be it known that I, **Mark J. Nixon**, a citizen of Canada, residing at 1503 Blackjack Drive, Round Rock, Texas 78681, have invented a new and useful **GRAPHICAL USER INTERFACE FOR REPRESENTING, MONITORING, AND INTERACTING WITH PROCESS CONTROL SYSTEMS**, of which the following is a specification. My legal representative for this provisional patent application is Mark G. Hanley (Reg. No. 44,736) of Grossman & Flight, 20 N. Wacker Drive Suite 4220, Chicago, IL 60606; phone 312.580.1020.

System Architecture

Table of Contents:

1	INTRODUCTION TO STARBURN.....	8
1.1	BACKGROUND	8
1.1.1	Operator Interface.....	8
1.1.2	Architectural	8
1.1.3	Organizational	9
1.2	VISION	9
1.3	OBJECTIVES.....	10
1.3.1	Operator Interface.....	11
1.3.2	Process Modules	12
1.3.3	Technical Architecture	12
1.3.4	Deployment.....	13
1.3.5	Development Practices	13
1.4	DEFINITIONS.....	14
1.5	REFERENCES	16
2	PROCESS GRAPHICS AND THE OPERATOR INTERFACE	17
2.1	OBJECTIVES.....	17
2.2	OVERVIEW	18
2.2.1	Static Graphic Components	18
2.2.2	Dynamics Elements	19
2.2.3	Process Performance Monitoring	21
2.2.4	"Smart" Process Graphics.....	23
2.3	REQUIREMENTS.....	24
2.3.1	Primary Operator Interface for DeltaV	25
2.3.2	Internet-accessible HMI for DeltaV	28
2.3.3	Wireless (Handheld) HMI for DeltaV.....	30
2.3.4	Display Configuration	31
2.3.5	Report Generation	33
2.3.6	Conversion of Process Graphics to Process Modules	34
2.3.7	Conversion of INtools to Process Graphics	36
2.3.8	Browse	36
2.4	DISPLAY TECHNOLOGY	36
2.4.1	Rich Display Client.....	36
2.4.2	Web-based Client	36
2.4.3	Smart Handhelds	37
2.4.4	Smart Phones	37
2.5	PROCESS GRAPHICS ARCHITECTURE	38
2.5.1	Process Graphics.....	38
2.5.2	Process Graphics Elements and Operations.....	39
2.5.3	Process Graphics Configuration	41
2.5.4	Process Graphics Display Format	48
2.5.5	Process Graphics Runtime User Interface	53
3	PROCESS MODULES	56
3.1	OBJECTIVES.....	56
3.2	OVERVIEW	57
3.2.1	Process Simulation	59
3.2.2	Expert.....	63
3.2.3	MSPC and PCA	65
3.2.4	Other Algorithms	67
3.3	KEY REQUIREMENTS.....	68
3.3.1	Process and Processing Elements	68
3.3.2	Streams.....	68
3.3.3	Process Algorithms	68

3.3.4	Process Modules	69
3.3.5	Documentation	69
3.3.6	Expert	69
3.3.7	MSPC and PCA	69
3.3.8	Other	69
3.3.9	Documentation	70
3.4	PROCESS MODULE ARCHITECTURE	70
3.4.1	Overview	70
3.4.2	Configuration	70
3.4.3	Runtime Execution	72
3.4.4	Operator Displays and Alarming	75
3.4.5	Diagnostics	75
4	RUNTIME WORKSPACE	76
4.1	OVERVIEW	76
4.2	OBJECTIVES	76
4.3	WORKSPACE DESKTOP MANAGEMENT	77
4.3.1	Dedicated and Controlled Desktop	77
4.3.2	Another Windows Application	78
4.4	INTERNATIONALIZATION SUPPORT	79
4.5	WORKSPACE FRAMEWORK	80
4.5.1	Framework Panels	80
4.5.2	Workspace Displays	83
4.5.3	Workspace applications	84
5	APPLICATION ARCHITECTURE	86
5.1	BACKGROUND	86
5.2	CONFIGURATION APPLICATIONS	87
5.2.1	Server-side Configuration Model	87
5.2.2	Client-side Configuration Model	90
5.2.3	Client Model Schema	91
5.2.4	Client Model and Object Cache	92
5.2.5	Command Framework	93
5.2.6	Configuration Framework	94
5.2.7	Package	95
5.2.8	Views	95
5.3	FUNCTIONAL BREAKDOWN OF APPLICATIONS	96
5.3.1	Controls	97
5.3.2	Views	97
5.3.3	Browser	97
5.3.4	Download	98
5.3.5	Drag/drop	98
5.3.6	Cut/copy/paste	98
5.3.7	Other UI commands	98
5.3.8	Application	98
5.4	CONFIGURATION FRAMEWORK	98
5.4.1	Overview	98
5.4.2	Assembly Dependencies	100
5.5	BROWSER FRAMEWORK	102
6	TECHNICAL ARCHITECTURE	104
6.1	BACKGROUND	104
6.2	DELTA V CONCEPT FOR SERVICE-ORIENTED ARCHITECTURE	105
6.3	SERVICES	106
6.3.1	Discovery	106
6.3.2	Session Service	109

6.3.3	Runtime Services	111
6.3.4	Database Services	112
6.3.5	Version Control Service	115
6.3.6	Historian Service	117
6.3.7	Historian Scanner Service	118
6.3.8	Alarms and Events Services	119
6.3.9	OPC UA Data Services	120
6.3.10	XML Files Service	121
6.3.11	HTTP Link Services	121
6.3.12	Server versioning	122
6.3.13	Client interface to server	122
6.4	A SERVICE EXAMPLE	122
6.4.1	Framework	123
6.4.2	Database Services	125
6.4.3	Servers	126
6.4.4	Service Clients	126
6.5	PROOF OF CONCEPT	127
6.6	TOPOLOGIES	127
6.7	KEY TECHNOLOGIES	128
6.8	VERSIONING	128
6.9	DEPLOYMENT	129
6.10	SESSION	129
6.11	SECURITY	129
6.12	GLOBALIZATION AND LOCALIZATION	129
6.13	USABILITY GUIDELINES	129
7	PROJECT ARCHITECTURE	130
7.1	OVERVIEW	130
7.2	CONCEPTS	130
7.3	SOFTWARE CONFIGURATION MANAGEMENT AND VERSION CONTROL	131
7.3.1	Software Configuration Management	131
7.3.2	Versioning	132
7.3.3	SCM System Layout	133
7.4	DEPENDENCY MANAGEMENT	134
7.4.1	Dependency Model	134
7.4.2	Development Environment	135
7.4.3	Build Environment	136
7.5	TESTING	138
7.5.1	Unit Testing	138
7.5.2	Regression Testing	138
7.6	PROJECT DOCUMENTATION	139
7.7	PROJECT MANAGER MODEL	140
8	DEVELOPMENT PRACTICES	143
8.1	COMPONENT AND SERVICE BASED DEVELOPMENT	144
8.1.1	Layering and Stereotyping	145
8.1.2	Component Management	145
8.2	ARCHITECTURE VERSUS SOLUTIONS	146
8.3	REQUIREMENTS	146
8.4	ANALYSIS	148
8.5	DESIGN	148
8.6	IMPLEMENTATION	149
8.7	VALIDATION AND VERIFICATION	149
8.8	SAMPLE DIAGRAMS	151

8.8.1	Component Diagram.....	151
8.8.2	Component Sequence Diagram	152
8.8.3	Use Case Diagram.....	152
8.8.4	Collaboration Diagram	154
8.8.5	State Diagram	155

Table of Figures:

Figure 1. Example operator graphic display that utilizes 3D components	19
Figure 2. Faceplate Accessed from a Control Dynamo.....	20
Figure 3. Example Hierarchy for Display, Trend and Information Access.....	21
Figure 4. Process Performance Included in Operator Graphics	23
Figure 5. Starburn Applications Architectural	24
Figure 6. Interaction of Process Graphics with Process Modules.....	34
Figure 7. Rich Client	36
Figure 8. Web Client	37
Figure 9. Smart Handhelds.....	37
Figure 10. Smart Phones.....	38
Figure 11. Process Graphics Editor.....	43
Figure 12. Using Process Equipment Pallet.....	44
Figure 13. Using Measurement Pallet	45
Figure 14. Special Pallet.....	46
Figure 15. Display Reference	46
Figure 16. Engineering and Maintenance Views.....	47
Figure 17 – Process Graphics Display XAML File Structure	48
Figure 18 – Display Section.....	48
Figure 19 – Dynamics Section.....	49
Figure 20 – Sample Dynamics Definition	50
Figure 21 – Dynamics Instance Structure	51
Figure 22 – Event Dynamics Structure	52
Figure 23 – Object Collaboration Diagram for Loading a Display	53
Figure 24 – Object Collaboration Diagram for Dynamics Processing	54
Figure 25. Process Graphic.....	57
Figure 26. Process Graphic – Advisory Alarms.....	58
Figure 27. Fluid and Gas Flow Simulation – Specified Starting Pressure.....	62
Figure 28. Fluid and Gas Flow Simulation – Vessel Establishes Starting Pressure	62
Figure 29. Fluid and Gas Flow Simulation – Pump or Fan Establishes Starting Pressure	63
Figure 30 – DeltaV Expert Overview	64
Figure 31. Interaction of the PCA, Expert kernel with Process Modules.....	67
Figure 32. Configuration Environment.....	71
Figure 33. Integrated Control Environment	73
Figure 34. Server-side Configuration Model.....	87
Figure 35. Published Server Model	88
Figure 36. Query Model	89
Figure 37. Client-side Configuration Model	91
Figure 38. Mapping Server-side model to Client-side model	92
Figure 39. Client Model	93
Figure 40. Command Framework.....	94
Figure 41. Configuration Framework.....	95
Figure 42. Application Architecture – Shown with DeltaV Explorer.....	96
Figure 43. Configuration Framework Layering	99
Figure 44. Package Dependencies	101
Figure 45. Implementing System-specific Services.....	106
Figure 46. Discovery Service.....	107
Figure 47. Discovery Service - Register Service.....	108
Figure 48. Discovery Service - Locate Service	109
Figure 49. Session Service.....	110
Figure 50. Runtime Services	111
Figure 51. Configuration Architecture.....	113
Figure 52. Database Services	114
Figure 53. Version Control Server	116
Figure 54. History Server.....	117
Figure 55. History Scanner Service.....	119
Figure 56. Service Architecture Framework	123
Figure 57. Database Services	125
Figure 58. Service Client	126

Figure 59. Configuration Models	141
Figure 60. Runtime Models	142
Figure 61 Component Diagram	151
Figure 62 Component Sequence Diagram	152
Figure 63 Use Case Diagram	153
Figure 64 Collaboration Diagram	154
Figure 65 State Diagram	155

1 Introduction to Starburn

This document describes the Starburn Technical Architecture. The Starburn Technical Architecture is a collection of architectural improvements, features, and applications that both integrate into and extend the existing DeltaV System. The architectural improvements address issues related to multiple site development, product releases, versioning, languages, scalability, extensibility, performance, and broadening the range of target UI devices. The architecture is being designed to more fully support new features and applications related to Abnormal Situation Prevention, Improved Alarming, Simulation, and Improved Control. All of these architectural improvements and features come together to provide a completely new Operator Interface and Runtime Workspace.

This document describes the architectural enhancements required to realize the objectives of the Starburn program.

1.1 Background

1.1.1 Operator Interface

The current DeltaV System includes Intellution's iFIX software integrated in as the DeltaV Operator Station. The software provides both display configuration as well as a runtime workspace. The runtime workspace is able to integrate DeltaV applications as well as 3rd party applications. The current DeltaV Operator Interface Software has been successfully used to integrate DeltaV, PROVOX and RS3. It is also currently being used to integrate Bailey, Teleperm, and most recently, Honeywell operator functions. The current Operator Software features are well described in the user documentation available with DeltaV.

Since the original vision for DeltaV there have been many innovations. On the Operator Interface front there is shift towards scalable graphics, "smart" process elements, integrated help, and application integration. There is an expectation that Operator Interfaces be able to run in either dedicated or non-dedicated mode as well as in thin client mode. With the recent explosion in UI technology there is also the expectation that Operator Interface capabilities be available on a wide range of devices. All of these interfaces need to be robust and secure.

The vision for these features is described in the following document:

`$/Hawk/Starburn/Vision/HLT Starburn SUI.doc`

1.1.2 Architectural

The original design of DeltaV was heavily biased by the object-oriented technologies available at the time the architecture was created. The promise was that heavy use of inheritance combined with other OO features would increase code sharing and in-turn reduce the amount of code written. The dilemma this creates is this - although heavy use of inheritance and code sharing reduces coding effort, it increases dependencies. These dependencies extend all the way through the build and release images. An example should help to illustrate this.

In the current DeltaV system applications and supporting database and runtime services are all tightly bound together. On the configuration side all client applications are built on a common set of data components representing data in the database. Any change to the data components means that client applications must be rebuilt. These things make it difficult if not impossible to version applications independently of the database and runtime. It is also virtually impossible to move the development of any single application to another development site because of the interdependence of the client data components. The database, runtime, and system must be built as whole and released as a single install image.

The solution is to divide the system into loosely couple services with layered independent applications (OO techniques will continue to be used). Each service has clearly defined external interfaces that can be individually tested and versioned. Applications can be built independently. By drawing lines in the sand judiciously a balance between loose coupling and development effort can be found.

The downside is that there will be more duplication of code – in some cases the same classes will exist in several projects. This is a fair price to pay for the independence created between applications and development teams.

Another objective presented later in this section deals with regression testing. In the current system there is limited support for regression testing above the lowest levels in the system (e.g. database and control layers). The fact is that since the user interface is tightly bound in with the client data components any such testing would be extremely difficult to maintain. In the architecture described here all of the interfaces should be able to be fully tested.

1.1.3 Organizational

Other factors are also at play – the development group that created iFIX is no longer part of the Emerson Process Management group. To offset this as well as address the large increase in scope described in the vision document a new development group in Manila has been created.

At the same time the development group in Pittsburg has a strong interest in utilizing several of the developments to enhance Ovation. In return developments from that group should be able to be integrated with the core DeltaV System.

1.2 Vision

Starburn includes a combination of enhancements to the basic operator environment as well as enhancements to the control infrastructure to support new predictive functions. The vision begins with an overview of current products and then adds in new Starburn features.

Process Control Systems are made up of many components. At the plant floor level are controllers, field devices, various IO interfaces, digital and/or combined analog/digital buses, etc. The field devices and bus devices, for example, valves, valve positioners, switches and transmitters (e.g., temperature, pressure, level and flow rate sensors), perform process functions such as opening or closing valves, measuring process parameters, etc. Newer devices tend to be smart field devices. These smart field devices in addition to measuring signals and controlling final elements, can also perform control calculations, alarming functions, as well as other functions. Dedicated controllers in this model both execute control strategies and coordinate control strategies executing on field buses.

Information from the field devices and the controller is made available over a data highway to operator workstations, data historians, report generators, centralized databases, etc.,. These operator workstations run applications that enable an operator to perform functions with respect to the process, such as changing settings of the process control routine, modifying the operation of the control modules within the controller or the field devices, viewing the current state of the process, viewing alarms generated by field devices and controllers, simulating the operation of the process for the purpose of training personnel or testing the process control software, keeping and updating a configuration database, etc.

A configuration database, which resides in the ProPlus, enables users to create or modify process control modules and download these process control modules via the ACN to dedicated distributed controllers. Typically, these control modules are made up of interconnected function blocks, SFC's, PLM's, Equipment Modules, Units, etc which perform functions within the control scheme based on inputs and which provide outputs to other function blocks within the control scheme. As part of the configuration function users create/change process graphics. These process graphics run inside a dedicated runtime workspace or in some cases, containers that load into internet explorers.

Process Graphics usually work together with other applications such as Alarm Banners, Alarm Summaries, Trending, etc to provide a complete operators interface. These operator display applications are typically implemented on a system wide basis. They typically execute in one or more workstations. Both pre-built displays and customer configured displays are used to provide Operations, Maintenance, and other groups with information about the operating state of the process, process equipment, and devices. Typically, these displays take the form of alarming displays that receive alarms generated by controllers or devices within the process plant, control displays indicating the operating state of the controllers and other devices within the process plant, maintenance displays indicating the operating state of the devices within the process plant, etc. These displays are generally preconfigured to display, in known manners, information or data received from the process control modules or the devices within the process plant. Often displays are created through the use of objects that have a graphic associated with a physical or logical element and that is tied to the physical or logical elements to receive data about the physical or logical elements. The object may animate the graphic on the display screen based on the received data to illustrate, for example, that a tank is half full, to illustrate the flow measured by a flow sensor, etc. While the information needed for the displays is sent from the devices or configuration database within the process plant, that information has traditionally been used to provide a display to the user containing that information. As a result, all information and programming that is used to generate alarms, detect problems within the plant, etc. must be generated by and configured within the different devices associated with the plant, such as controllers and field devices during configuration of the process plant control system. Then is this information sent to the operator display for display during process operation.

While error detection and other programming is useful for detecting conditions, errors, alarms, etc. associated with control loops running on the different controllers and problems within the individual devices, it is difficult to program the process plant to recognize system-level conditions or errors that must be detected by analyzing data from different devices within the process plant¹. Operator displays have typically not been used to indicate or present such system-level condition information to operators or maintenance personnel. There is currently no organized manner of detecting certain conditions within a plant, such as flow conditions and mass balances, as materials move through a plant, much less an easily implementable system for performing these functions on a system-level basis. To address these limitations process and processing elements are being introduced into the DeltaV System.

The execution environment for process elements is the Process Module. Process modules include behavior or methods, also called process algorithms that can be used to simulate and detect process conditions. Each process element is associated with a plant entity, such as a field device or some logical element, and includes a data store for parameters or variables associated with that entity. The process elements are coupled to the entity, either directly or through an external reference, to receive data associated with that entity. Each process element can also be coupled to other process elements within the Process Module environment to send data to and received data from the process, process equipment, and devices. For example, a process element for a tank may be coupled to process element for pumps or flow transmitters upstream and downstream of the tank and receive data indicative of the upstream and downstream flows into and out of the tank. A method associated with the tank object may detect or even predict abnormal conditions in the tank by, for example, comparing the level of the tank with the expected/simulated level in the tank based on the flows into and out of the tank. Process modules may include flow algorithms that can be implemented on the combination of entities to detect system-level conditions, for example, to detect mass balances, gas blending, flow conditions, etc.

1.3 Objectives

Starburn is collection of features, applications, and architectural improvements. Because of the magnitude of the development program development practices and procedures are also affected. The objectives for the program are described in the following sections.

¹ Or remotely connected IO (i.e. Remote IO) and Controllers (i.e. Brick).

1.3.1 Operator Interface

The Operational Interfaces is being designed to run in both dedicated and non-dedicated mode. It will run as a rich client or as part of a browser style interface utilizing web services. It is available on workstations, laptops, tablet PC's, handhelds, and smart phones. These objectives are summarized below:

- 1- Dedicated mode - Operator workspace supports a dedicated mode for use in control rooms or any place where the display arrangement needs to be fixed and/or access strictly controlled (i.e. Kiosk style application environment).
- 2- Non-dedicated mode - Operational workspace supports a non-dedicated mode for use by configuration folks, engineers, plant management, and others where flexibility and the richness of the interface are more dominant features.
- 3- Support multiple UI devices.
 - a. Operational interface supported on a wide range of user interfaces – including Rich Clients, Web Browsers, Handhelds, and Smart Phones.²
 - b. Support for new types of features such as integrated Voice and Video, new input techniques such as those used in Tablets.
- 4- Data Services.
 - a. Support real-time data services, embedded historian, alarms & events, etc.
 - b. Support for external data services – specifically OPC.
 - c. Support reading from XML files.
 - d. Support access to other service interfaces (may be disabled in secure mode).
- 5- Display Management and Reuse
 - a. Process Graphics will be able to be able to utilize composite structures.
 - b. Displays can be set up and managed as part of the Class-based control hierarchy.
- 6- Integration of Control, Alarming, and Abnormal Situation Management
 - a. Integrated Batch Operator Interface.
 - b. Integrated Advanced Control Operator Interfaces.
 - c. New applications – Route Management, Efficiency Calculations, Optimizations, Mass balance, etc.
 - d. Integrated Abnormal Situation Prevention.
- 7- Integration of 3rd Party Applications
 - a. Improved integration across Applications.
 - b. Decoupling of applications from the DeltaV core subsystems.
- 8- Support multiple DCS's
 - a. Support real-time data from multiple systems, e.g. DeltaV and PROVOX.
 - b. Support alarming from multiple systems, e.g. DeltaV and PROVOX.
- 9- Trending
 - a. Full support for reading from DeltaV Historian.
 - b. Full support for writing to DeltaV Historian – specifically Lab Data Entry.

² Will become the new Operator Interface.

1.3.2 Process Modules

10- Improved Control

- a. Mass Balance.
- b. Energy Balance.
- c. Simple Composition.
- d. Custom Calculations.

11- Abnormal Situation Prevention

- a. Using improved control calculations generate alert messages.
- b. Integrate specialized calculations from other Emerson Process Management divisions.
- c. Integrate specialized calculations from 3rd parties or customers.

12- Reduced engineering efforts

- a. IO and Process checkout.
- b. Elimination of duplicate display and simulation configuration when engineering systems with simulation requirements.

13- Framework for Operator Training

- a. Process Graphics that can be used for process simulation and process calculations.
- b. Process Graphics include dynamics from both Control Environment as well as Process Modules.

14- Improve integration with High Fidelity Simulation Applications

- a. Interface for exchanging runtime configuration. Included in the exchange will be parameter reference information and data type information.
- b. Interface for exchanging Operator Commands.

1.3.3 Technical Architecture

15- Support Versioning - ship versions of applications separate from core system³.

16- Deploy hot fixes and service packs.

17- Common usability guidelines across all applications and development sites.

18- There are now four development sites around the world (Austin, Leicester, Pittsburg and Manila). The aim is to allow teams at these locations to pursue independent development within the cycle of DeltaV releases. It should be possible to move the development of parts of the system to a different geographical location and not introduce source code dependencies between the sites.

19- Future releases of DeltaV will comprise of a core system and a suite of applications. The core system will be presented in the form of a series of services. It will be possible to release versions of the applications independently to the core services. Applications and services must interoperate between versions. Applications must connect to services that are at most two versions behind and services must continue to support applications that are two versions behind.⁴

20- Partition functionality into functional groups.

³ We will need to build-in comprehensive support for our call center and service staff for them to cope effectively with this.

⁴ Additional level of system testing will be necessary to ensure all the versions that are required really do work together.

- a. Function groups need to be kept separate so that we can develop independent and potentially move things around.
 - b. Example – Browser is a functional grouping. A Browser framework is provided. All applications implement their own browsing support by implementing the interfaces provided as part of the Browser Framework.
 - c. structure of the commands DLL
 - d. Structure of drag and drop (some kind of Type/Handle base). Need a common system-wide spec.
 - e. copy/paste
- 21- Run or transfer several of the applications to Ovation
 - a. Process Graphics needs to run on top of Ovation (also need Browser Framework).
 - b. History needs to run on top of Ovation.
 - c. Browser Framework needs to be transferable to Ovation.
- 22- Run several of the applications on top of competitive systems.
 - a. Operator Workspace
 - b. Advanced Control
 - c. Custom Function Blocks
- 23- Localization separated out from builds (base build, localize using a toolkit – needs sophisticated tools than can allow dialogs to be re-worked to accommodate differences in languages).
- 24- Be able to move projects between sites.
- 25- Every service and application needs to have full regression tests built as part of the development. Every developer is responsible for developing tests for the test rig.
- 26- Native support for Longhorn service architecture and scalable graphics.
- 27- Support a wide range of UI devices.

1.3.4 Deployment

- 28- Full install off of DVD
- 29- Layered applications
- 30- Service packs
- 31- Hot fixes
- 32- On-demand deployment

1.3.5 Development Practices

- 33- Support for several releases being done at the same time
 - a. Starburn
 - b. Safe
 - c. Zones
 - d.
- 34- Austin performs a base build incorporating database code from Leicester. Other sites develop applications and extensions.

35- Localization can be performed after the core system is built.

36- DeltaV can be released in several manners

- a. New system from mfg installed with DVD or install image
- b. Applications released after the base system – don't require service upgrades
- c. Functional enhancements
 - i. update the database
 - ii. new runtime service
 - iii. new configuration application
 - iv. new displays
- d. Bug fixes

1.4 Definitions

Term	Definition
Service	<p>A reusable segment of business functionality which supports a significant portion of an important business process. A service is an interface offered by a component – its service interface is comprised of the service methods which together form the protocol of the service.</p> <p>Individual service methods are rarely used in isolation. A single use case may combine the service methods of several services within its interaction to provide the functionality described.</p>
Framework	A class hierarchy providing common behaviors and properties that can be inherited for specialization.
Pattern	A recurring theme in design, implementation or organization that is known to be effective and hence is documented so that it can be reused.
Component	Components are a vehicle by which services can be implemented.
Graphical IF	Any Graphical User Interface that participates in the display creation and viewing using the tools described in this document.
Operator IF	Graphical User Interface specifically designed to support Operator Functionality. Includes Alarm Summaries, Alarm Banner, Faceplates, and Dynamic behavior as is used today by current DeltaV users.
Web-based IF	Graphical User Interface that runs inside Internet Explorer. Currently the approach is to use XAML.
Smart Handheld	Graphical User Interfaces such as Compaq's IPAQ.
Smart Phone	Text and Graphical User Interfaces that run on new Smart Phone platforms.
Process and Processing Elements	Tagged graphical items that can be referenced by other configuration items as well as the runtime. They are normally equipment items such Pumps, Valves, Tanks, Reactors, etc.
Smart Process Links aka Streams	Smart Process Links are also referred to Streams in other simulation and optimization packages. Streams can have compositions associated with them. When they compositions associated with them the compositions flow along the stream through the Process Elements.

	Streams can be referenced by the runtime. They are most often associated with Pipes.
Process Algorithm	Similar to existing algorithm types, Process Algorithms are defined for generalized use for certain kinds of problems (e.g. FBD's are designed for continuous loops). New algorithm types include Mass Balance, Routing, Efficiency, Optimization, Economic Calculations, other.
Process Module	Have special properties including, Mode, Status, and Alarm Behavior. Can be assigned to workstations. Execute process flow algorithms.
Process Database	Stores information about Process Elements, Streams, Algorithms, and Modules. The database will allow users to define multiple views against the same database items – for example Several Process Graphics and Algorithmic views referring to the same set of Process Equipment.
Display Management	Manage graphical displays from within the DeltaV configuration system. This includes display assignment for download purposes, download status indicators, and version control.
Display Classes	Associating display classes with module classes allows displays and binding tables to be defined just once on a Module Class basis. Binding tables for each Module Instance created from that Module Class would be created automatically.
Graphics Primitives	These are built into the display configuration model along with their behavior. These include lines, polylines, ellipses, arcs, rectangles, text, images, composite (group). Each primitive in the editor has an XAML equivalent (although one editor primitive may be translated into multiple XAML objects, e.g. multi-line text, gradient filled objects etc.) User variables can be associated with any primitive. Variables are used by dynamic behaviors to control the algorithm and how items are displayed.
Graphics Group (composite)	Graphics primitives and other groups can be grouped together to form a grouped item which can behave as a single entity for editing purposes. User variables can be associated with any group. Variables are used by dynamic behaviors to control the algorithm and how items are displayed.
Component Primitives	Pre-built components that are built by EPM and distributed as components. Component Primitives have built-in behavior which can range from simple to complex. Component primitives have behavior which allows them to be configured. These components are for providing specialized interactions or controls. E.g. Trend controls, advance control interactions etc.
Process Graphic Item (aka - Library Item, either Template or Class)	Are pre-built single primitives or groups, including any defined variables and dynamic behavior. We will provide libraries of composites for common control equipment such as PUMPS, VALVES etc. Users can also create their own library of composites. Library items can be treated as templates, in which case a copy of the item is made if it is placed on a display, or classes in which case a link to the library item is made if it is placed on a display. When an instance of a class is created on another display the user can override the public parameters defined in the class, together with its viewport characteristics (position, size and rotation).
Display	A display is a graphics group which is capable of being displayed independently. It includes the definition of the "canvas" on which the display is drawn (its size, background color or background image). Public variables defined for a display can be used as parameters when the display is called up. For example, the color of pipework for a

	display could be controlled by a variable PIPEWORK_COLOR. When the display is called up the parameter PIPEWORK_COLOR could be set to blue.
Display Class	A display class is extremely similar to a display except that one or more of its public parameters are used as an alias in process value paths. For example, a display class could have process value paths including the alias MODULE (e.g. <code>##MODULE#/ATTR.FIELD</code>). MODULE itself is a parameter of the display. When the display is called up the parameter MODULE could be set to FIC-101A.
Display Template	Is exactly the same as a display except that it is placed in the library rather than associated with a particular equipment item, module, or item of hardware. A display template can be used as a starting point for creating displays.
Configuration Application	An application (.EXE) that is used to configure DeltaV. Examples of applications are: Explorer, Control Studio, and Recipe Studio. In general they are stand alone applications. Meaning that a DeltaV user launches an application directly without having to first launch a workspace and then choose to load an application.
Configuration Framework	Collection of libraries, services and interfaces used by the Application developers to create a DeltaV configuration application.
Docking Panel	Docking panels are rectangular display areas which host a single application view. A docking panel provides functionality similar to that found in visual studio. This means that a docking panel has implicit behavior such as: <ul style="list-style-type: none"> ○ Dock/undock ○ Auto hide ○ Drag/drop to other panels ○ Layout is persisted as user preference
DataSource	Abstraction of the data model that can interfaced to Client Models. Designed to work seamlessly with DataItems.
DataItem	Abstraction of the object that holds the handle or reference to the underlying data model, example the ClientObjectHandle for the ClientModel

1.5 References

2 Process Graphics and the Operator Interface

2.1 Objectives

The objectives for Process Graphics are:

- 1- **Integrated** – Process Graphic Displays is fully integrated with configuration, runtime, historian and other subsystems.
- 2- **Quick Edit/Runtime** – Users will be able to quickly switch to a run-mode to see how the display they are working on will look at run-time. Both configuration and runtime views can be run simultaneously⁵.
- 3- **Ease of use** – Process Graphics will be easy to use. Displays will be easy to call up on a wide variety of display devices. No display conversion is required to use displays on different devices. It will be easy to design displays for optimum usability on specific form factors.
- 4- **Best engineering practices** – The system will be designed to facilitate re-use of displays and sub-displays through both standard and user-defined libraries, display classes, etc, thus minimizing configuration and testing effort.
- 5- **Separation of the graphics from the scripting engine.** The graphics and it dynamics are defined in data that are animated using both scripts (C#) and supporting Class-based Assemblies. Scripting engines are part of the Longhorn-based operating systems, Browsers, etc.
- 6- **Flexible data bindings** – Process Graphic Displays can retrieve data from multiple data sources including Runtime, Historian, OPC, Services, XML Files, etc.
- 7- **Run on multiple systems** – The Process Graphics implementation can be used with both DeltaV and Ovation.
- 8- **Smart Objects** – Process Graphic Displays will fully support Smart Objects through Process Modules.
- 9- **Wide range of devices** – Runs on a wide range of devices – all the way from dedicated smart phones up through multi-screen consoles.
- 10- **Longhorn Look/Feel** – Designed from the ground up to take advantage of Longhorn Look/Feel.

⁵ The Process Graphics Editor will provide a quick run mode to test displays. This run mode will support most of the features available to the Process Graphics Runtime when run inside the Runtime Workspace.

2.2 Overview

The operator interfaces for process control continues to advance. Current operator interfaces take advantage of high resolution graphics and multiple screens. Some of the operator interfaces are also being made available through browser style interfaces.

Typically a graphic operator interface will consist of a series of process graphic displays that illustrate the pieces of equipment that make up the process. The primary advantage of such an interface is that more information may be presented to an operator in an efficient manner. However, the effectiveness of such interfaces depends greatly on the manner in which the graphic displays are structured in the system configuration.

2.2.1 Static Graphic Components

Soon after distributed control systems introduced support for graphic displays, an American National Standard was developed for Graphic Symbols for Process Display, ANSI/ISA-S5.5-1985. The stated purpose of this standard was to establish a system of graphic symbols for displays that are used by plant operators for process measurement and control. This was done to promote comprehension by the operators of the information that is to be conveyed through the displays, and to provide uniformity of practice throughout the process industries. The benefit of such standardization is a reduction in operator errors, faster operator training, and presentation of the process to facilitate use by the plant operators. This standard is followed by many companies and is suitable for use in the numerous industries. However, the standard's two dimensional line drawings have been made obsolete by advent of faster computers with graphics card supported by most system. High definition visual display units, VDU, are commonly available based on Cathode Ray Tube (CRT) or Liquid Crystal Display (LCD) technology that support 1280x1024, 1600x1200 or even higher resolution. Object orient software allows more complete 3D representation of process equipment.

Current control systems include a process graphic display editor and a comprehensive library of pre-built display objects with professionally drawn 3D representation of process equipment including pumps, valves, meters, piping, tanks, etc. Also, some systems may offer the capability to import graphics from a number of different sources including INtools, AutoCAD and Windows metafiles such as Visio vector drawings, as well as JPEG and BMP image formats. These components may be used to create a realistic representation of the process to which dynamic information may be added. An example of an operator display created using a library of 3D components is illustrated below.

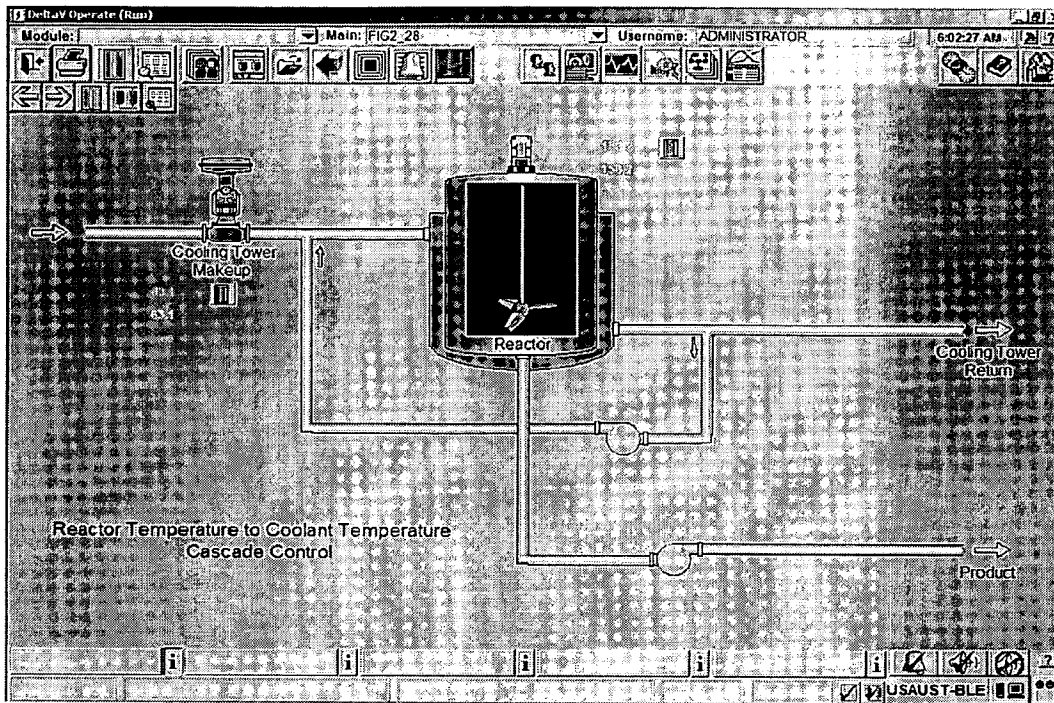


Figure 1. Example operator graphic display that utilizes 3D components

2.2.2 Dynamics Elements

Predefined objects for information access are a standard part of graphic tool sets. Basic components such as knobs, dials, slide bars, and buttons that may be dynamically linked to information within the control system. In addition, a library of custom dynamos will be typically provided that may be used to access measurements, calculation, and analog and control components within the control system. Often these pre-engineered dynamos may be customized to fit the specific needs and standard of a plant.

Dynamos typically provide the key parameters associated with measurement and control functions. For example, a dynamo used to display a control loop may display the control parameter, the setpoint and control output. Engineering units may be shown to provide context to a value displayed by the dynamo. The status of process alarms associated with the control may be reflected in the dynamo through color change e.g. background color of the control parameter value. Also, to eliminate clutter in the display, the fact that a loop is not in its designed/normal mode may be indicated through color change e.g. Units background color change. The ANSI/ISA-S5.5-1985 standard provides general guidelines on the color in a display e.g. Red is reserved to indicate an emergency situation such as Stop, highest priority alarm. The predefined dynamos provided with a control system may conform to this standard. However, these guidelines may not be consistent with the manner in which color has been used in a particular plant. In such cases, the colors used in dynamos may be modified to conform to the plant standard.

When the operator accesses a dynamo, then a faceplate may be automatically presented that provides further information on the parameter shown. For example, when a dynamo associated with a control loop is accessed, the faceplate presented to the operator may be used to modify the mode, change a setpoint or output. In addition, the capability may be provided to access further information such as loop tuning by selecting a detail display reference. In response, another window will be provided from which detail information about the control may be viewed. Which parameters in a detail display may be changed will be determined by the system security and the users authorization level. In some cases, operator access to information from a graphic display may be specified on a parameter by parameter basis. An example of a faceplate display generated by accessing a control loop dynamo is shown in diagram below. The associated detail display is also illustrated.

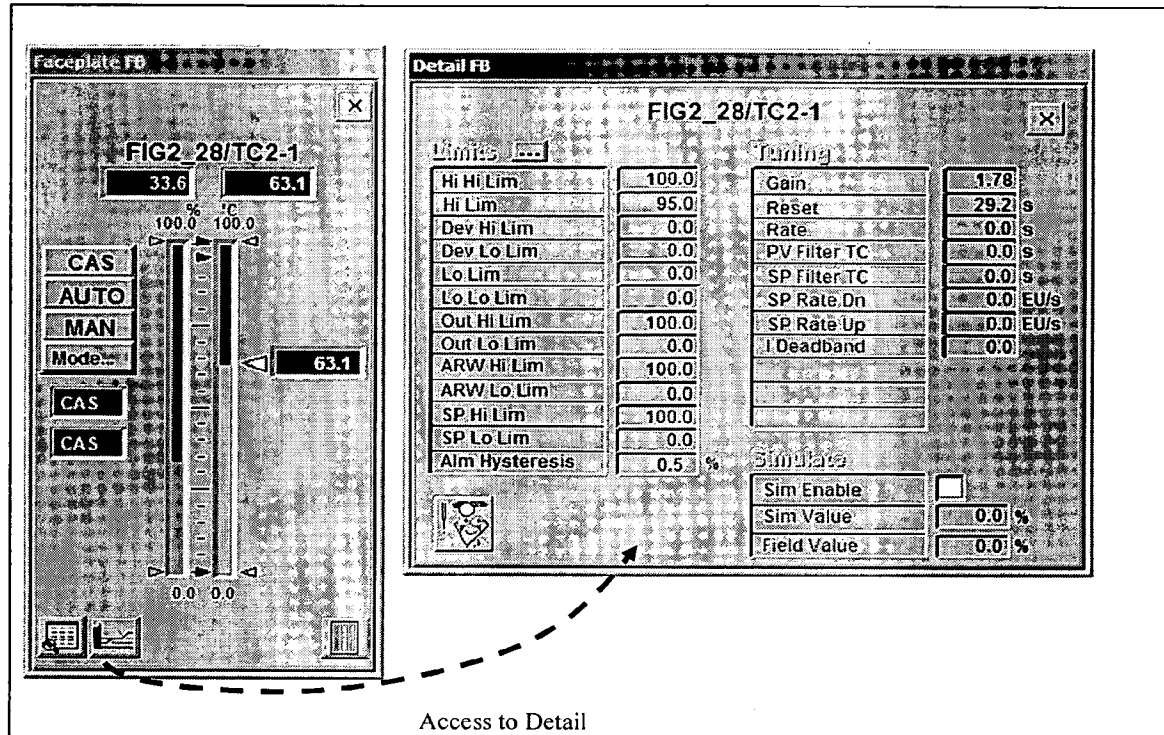


Figure 2. Faceplate Accessed from a Control Dynamo

Where the control system supports the use of aliases in the definition of similar pieces of equipment, the dynamic display components may also be designed to support dynamic referencing based on the piece of equipment selected for display. In such cases, pre-configured aliases and attributes may be used in place of an object tag or graphical attributes normally defined as part of the display object. Thus, aliasing supports a high degree of flexibility and re-usability since objects may connect to different I/O points and represent different graphical attributes, appearances, and security. Such capability may eliminate the need to re-building similar display objects.

The use of a mouse in the operator interface must be considered in the Process Graphics design. Displays that use this type of interface will typically be designed to support both horizontal (below the display) and vertical (to the right of the display) toolbars. The default toolbars to support the following functions will be provided as a standard part of the Process Graphics:

1. Time and date display
2. Alarm list with direct access to the display required to acknowledge the alarm
3. Navigation to alarm summary, main menu, system status, last display, page back and forward and print

4. Alarm acknowledge, and silencing

Using the page forward and page back of the display toolbar; it is possible for an operator to access displays that contain information upstream or downstream of the displayed process. In addition, dynamos may be provided to allow another display to be accessed. Through the use of these tools, it is possible to create a display hierarchy from which an overview display may be used to access the key display in each process area. Within the process area, the ability may be provided using page forward/backward toolbar to pan through the displays within the process area. Also, using a dynamo for display access and trend window access, it is possible to provide the operator with startup/shutdown information and historical plots as illustrated below.

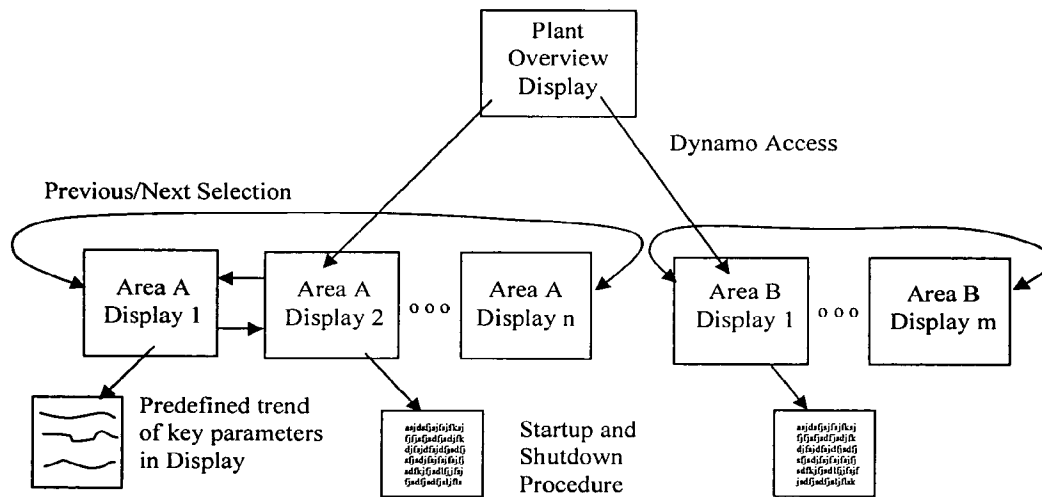


Figure 3. Example Hierarchy for Display, Trend and Information Access

The graphics editor supported by the control system may allow animation to be incorporated into a graphics display. Using this capability, the static graphic component for process equipment may be modified to indicate the status of the equipment e.g. motor on or tripped. Also, animation may be used to represent dynamic data associated with the equipment such as showing the level of a tank using filling technique or show the status of an agitator through display change to indicate motion. In newer products, layer controls (send-to-front, back, send-forward, backward) may be used to show different information in the display based on the state of the process operation.

2.2.3 Process Performance Monitoring

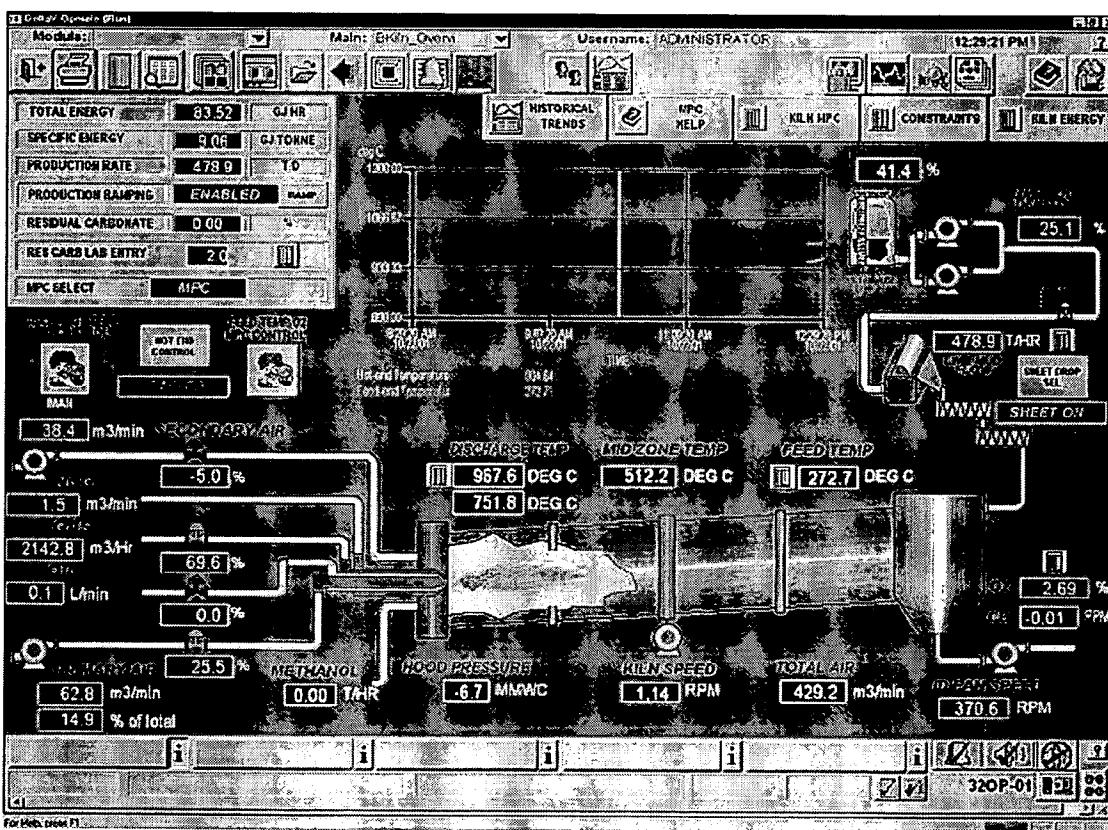
The graphic display capability of the control system may be used to create special displays to allow the status of critical equipment to be easily monitored. Some examples of these types of requirements are:

- First Out Indication on a process shutdown
- Vibration Monitoring
- Burner Management
- Sootblower

- Safety System Status

The associated displays can be structured to summarize the information. In some cases where moving equipment involves animation, for example a sootblower, animation may be effectively used to allow an operator to quickly access the operation⁶.

The calculation capability of most control systems may be used to implement on-line calculation of operation cost, efficiency, etc. This type of information may be easily incorporated into the operators' graphic display so that he can use this information to improve the process operation. An example of such types of calculations for a lime kiln operation is shown in the figure below.



⁶ Animation is an important part of Operator Graphics. In the case described here, a simple animation showing the blower spinning is an effective visual indication to the Operator that the equipment motor is running.

Figure 4. Process Performance Included in Operator Graphics

A variety of techniques may be used to integrate subsystem⁷ information in the control system. When this is done the standard graphics and dynamos may be used to create operator displays to allow access to this information. In some cases 3D plotting of matrix values is required to show information (e.g. sheet gauging information).

In special cases the information may need to be displayed in a manner that is not supported by the control system. To address these special requirements the graphics display needs to support terminal services.

2.2.4 “Smart” Process Graphics

The discussion so far has centered around the use of Graphics, Faceplates, Detail Displays, Trending, and other aspects of Operator Interfaces that have evolved over time and are available in some form in most control systems. In Starburn, the base that the graphics are built on and something we are calling “Smart” Process Graphics. So what are “Smart” Process Graphics?

The term “Smart” originates from the vision document. In that document the vision calls for a new generation of graphic elements that have behavior that far exceeds what is available today. In the vision document these “Smart” behaviors have knowledge about how process elements are connected, have built-in behavior corresponding to the process element they are associated with, and are able to work together to predict abnormal conditions that would otherwise lead to failures, loss of production, etc. The concept document also introduced the notion “Smart” documents. Starburn addresses both the traditional operator interface functions as well as the “Smart” process behaviors described in the vision document.

So what’s behind these “Smart” behaviors? How can displays which have a limited lifetime continue to provide this smart behavior even after they are unloaded from the system? What do displays have to do with smart documents? The answer to the smart behavior is that the Process Graphics are associated with process elements which in-turn execute inside a new kind of module called Process Modules. In this “Smart” environment several new types of process algorithms will be created. These process algorithms support new functionality for Gas Blending, Route Management, Efficiency Calculations, Optimizations, Mass balance, Simulation, etc. These new Process Algorithm types run inside Process Modules at run-time⁸. Process Modules will be discussed in the next section. Smart documents allow users to create documents that embed Process Graphics in them. By specifying a point in time these documents can extract data from the historians.

The relationship between process graphics, process modules, configuration, and supporting services is illustrated in the diagram below.

⁷ The Subsystem in this context refers to an equipment subsystem such as gauge, motor starter, vibration monitor, etc. In the example used here a sheet gauge is used to measure the properties across a material, for example paper on paper machine or plastic being extruded from an extruder. The cross direction and machine direction properties are ideally displayed as 3D Plot – width, thickness, and time providing the X, Y, and Z coordinates.

⁸ The Virtual Controller will be changed to run inside the .Net Managed Environment

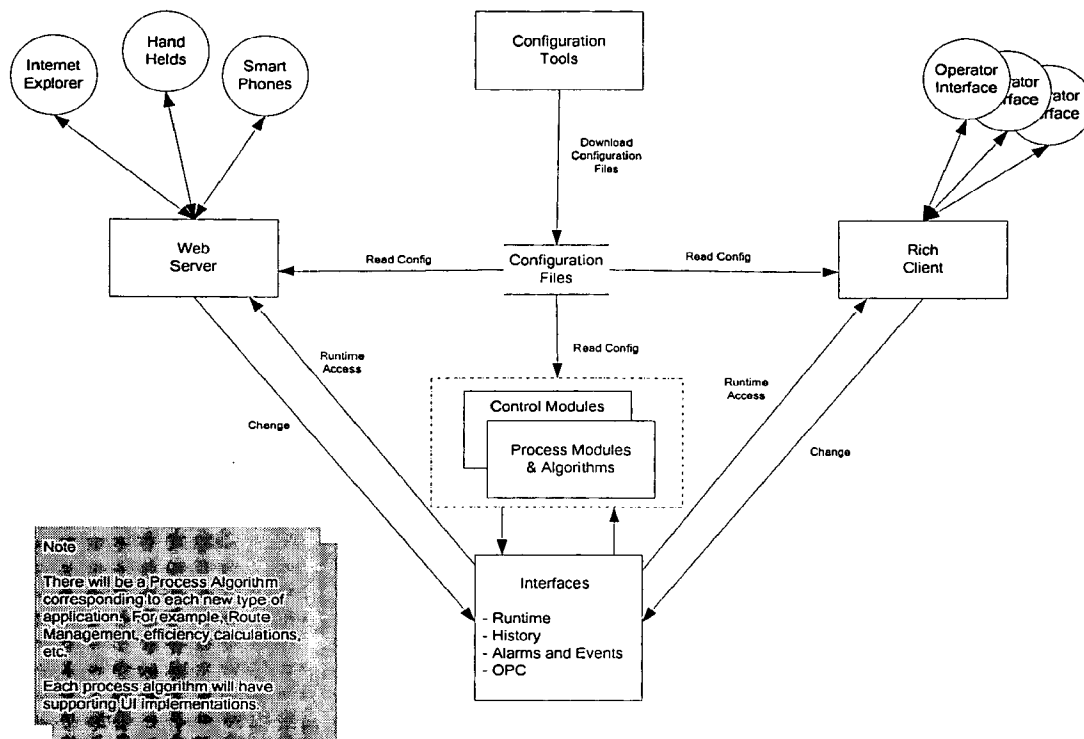


Figure 5. Starburn Applications Architectural

2.3 Requirements

The Starburn architecture for DeltaV focuses on a centralized representation (and database) for process plant objects, interconnections and layout information. The properties of these objects and interconnections are broad enough (and probably extensible enough) to support a number of DeltaV applications, including:

- Operator interface "graphic displays" and other types of HMI applications
- Process Module generation. The Process Graphics implementation needs to provide an interface and schema to support conversion from Process Graphics to Process Modules (the Process Module environment does the conversion and reconciliation).
- Report generation. The Process Graphics implementation needs to provide an interface and schema to support activation from a Report Generator. The Report Generator will be able to specify current time, an absolute time, or a relative time. When time is not current values will be selected from the DeltaV Embedded Historian.
- Highly automated "process rule" checking applications (e.g. mass balance calculations and detecting leaks, etc.). The Process Graphics system should be able to activate validation functions through the Process Modules verify functions.
- Coordinating the use of shared equipment (e.g. dynamic unit allocation, resource management for material transfers, etc.)

Particular attention is to be given to providing various HMI options for interacting with information from the DeltaV system, and from other systems providing popular well-defined interfaces (e.g. OPC interfaces). The range of HMI options are to include:

- Rich-featured HMI, tailored to DeltaV, suitable as (our premier example of) a primary operator interface.

- Intra-net connected "remote", thin-client⁹ primary or auxiliary operator interfaces for DeltaV and other compatible systems.
- Internet accessible "casual" (may be view only) HMI applications for DeltaV run-time information.
- Specialized HMI applications involving wireless connections and portable devices (handheld, tablets, etc.) tailored to take advantage of the specialized features (and working within the limitations) of those devices.

For users who employ several of these HMI options, the goal is to significantly minimize their engineering effort; taking maximum advantage of the centralized database of plant objects, connections, and layout information for use with the various HMI options.

...and better integrate the operator functions currently provided by:

- DeltaV Logon
- DeltaV "desktop"
- DeltaV Batch Operator Interface
- DeltaV MPC Operate
- Diagnostic Explorer
- Process History View
- Batch History View
- (AMS) FF Device Details application
- :

...and with new inter- and intra-net remoting features, largely remove the need for:

- DeltaV Remote Operator Stations
- DeltaV Web Server

2.3.1 Primary Operator Interface for DeltaV

Ultimately the Starburn HMI architecture must support the premier, full featured, high performance, primary operator interface for the DeltaV system. Key requirements in this scenario include:

2.3.1.1 Performance

Promoted as the primary operator interface for DeltaV, we need this variation of the Starburn HMI architecture to be optimized for the best possible (display building) performance. This suggests:

- 1- High degree of parallelism between data gathering and display rendering computations.
- 2- Display configuration content optimized to help the data gathering problem.
- 3- Minimizing server layers (e.g. working directly to ROC rather than OPC for DeltaV parameters) and RPC hops.
- 4- Top to bottom optimizations (e.g. "parameter list" oriented request, not blocking on resolving EU ranges, etc.)
- 5- Set of pre-engineered displays¹⁰ optimized for speed (e.g. avoid costly display objects, minimize animations, etc.)

2.3.1.2 Dedicated Operator Interface Workstation

We need to continue to support those who want a locked-down, "captive" operator interface application as the ONLY application accessible (to most users) on a workstation¹¹.

⁹ Here "thin-client" refers to any technology which substantially reduces the deployment and maintenance cost/effort involved in providing a HMI.

¹⁰ The ones we use for benchmarks, or for display performance-sensitive customers in the demo room.

¹¹ The Runtime Workspace provides much of the "locked-down" and "captive" behavior.

- 6- The HMI application has options to make it be automatically started after the workstation boots.
- 7- The HMI application has options to allow the user (with appropriate DeltaV security keys) to restart the workstation. The workstation restart process doesn't make it possible to use other applications.
- 8- The HMI application has options to prevent the user from "escaping" or terminating this application.
- 9- The HMI application does not provide a means for the user to generally manipulate files/directories (e.g. the standard file browser dialog).
- 10- The HMI application has options to allow the user (with appropriate DeltaV security keys) to start certain other applications (presumably also DeltaV applications designed to cooperate in the "captive" operator interface architecture.)
- 11- In the extreme case, it is possible for the HMI application on a workstation to be configured so that the HMI application (and any child applications that it manages) is accessible, even through workstation reboot operations.

2.3.1.3 Specialized Operator "Console" Hardware

We should continue to expect user requirements for:

- Supporting "multiple-head" operator stations, permitting more information to be on display (and more display management issues).
 - Utilizing function keys and/or auxiliary key pads to perform (user configurable) frequently used HMI interactions.
 - Utilizing touch screens in addition to (or instead of) a mouse.
- 12- The tools for defining HMI displays must allow for displays sizes and aspect ratios suitable for "multiple-head" operator stations.
 - 13- Users wishing to avoid display window placement and sizing actions for their operators should be able to define the HMI display "framework" for their "multiple-head" operator station. The HMI application uses the "framework" to position (and optionally scale) displays when they are built¹². Default frameworks (for typical 2 and 4 head monitor configurations, optimized for the pre-engineered displays) are provided.
 - 14- The HMI application provides a means for the operator move/pin displays in the framework, so that he can easily control the combination of display appearing at any time.
 - 15- Specialized frame properties can be set:
 - Batch operations display target
 - Process/Batch history view target
 - Unit/equipment module alarm summary target
 - Faceplate "bag" (dynamically selected set of N faceplates)
 - :

2.3.1.4 Using Engineering Drawings for Displays

Many "process graphic" operator displays are derived from existing drawings used to design the physical plant and instrumentation system. Users face a number of barriers to utilizing these drawings (to decrease the engineering effort) for operator displays:

- The size and aspect ratio for the operator displays do not align very well with plant drawings. The applications that render the operator displays may not have very powerful tools for dynamically adjusting scale, zooming or panning, so the drawings end up being unreadable or too difficult use.

¹² Frames in the framework can be configured with "auto-load" display, "do not change" and "do not obscure" properties, so traditional alarm banner and button banner type behavior is possible.

- Creating useful operator displays from a drawing (e.g. a P&ID) generally involves the addition of many dynamic elements (digital readouts, bar graphs, selection/action targets, display navigation buttons for off page connections, etc.) If the operator display engineering tools cannot "understand" these drawings as anything more than bitmaps, they cannot offer much help to accelerate this process.
- P&ID drawings often have content that is not always useful in operator displays (e.g. manual control valves, instruments for manual readings, legend and description blocks, even the instrumentation symbols and connections may be more clutter than a well placed 1/2 size faceplate).
- Managing the information (esp. the dynamic element) density is often the major design issue in creating operator displays, while it may not have been a consideration in the creation of the existing plant drawings. Therefore if the resulting operator display would be too dense (or not dense enough), the "drafting" work invested in the plant drawings may be difficult to reuse.

To help overcome these barriers, it suggests:

- 16- We need "plant drawing importers" that preserve a richer data model of what the drawing represents. Presumably we should focus on working with the leading plant design packages (or at least the ones that would benefit Emerson Process Management service providers.)
- 17- For systems/users who don't need a leading (expensive) plant design package, consider spinning off a version of the Starburn engineering tools that could be used to create P&ID drawings¹³.
- 18- To deal with the content coming from plant drawings that may not always be desired for an operator display, it suggests a means to segregate and sometimes hide some of this content. It would seem that the "layers" concept useful in CAD type drawings might be applied here as well:
 - Different types of plant drawing content would be assigned to different layers of the display.
 - The display engineer can add new display content (usually dynamic elements) to other layers of the display. He can then choose which layers will be included in the final display definition, and which layers show up by default (and which are included but hidden until an action is taken to make them appear).
 - A means in the HMI display rendering software (an advanced feature) letting the user see which layers are available, and to hide/expose the ones they choose.

Net: the same display definition can be used for a display intended for "operating" (lots of dynamic elements) and a advanced user can manipulate layers to get a rendering that is as close as possible to the original plant drawing.

- 19- The display engineering tool relates tagged objects in the plant drawing (e.g. a FC-101 flow controller symbol) with a control module of the same name defined in the DeltaV configuration, and provides a means to easily add groupings of dynamic elements¹⁴ showing the most important parameters of the control module.
- 20- Similarly, for Process Modules, the display engineering tool relates tagged objects in the plant drawing with a process module of the same name defined in the DeltaV configuration, and provides a means to easily add groupings of dynamic elements showing the most important

¹³ Encouraging users to start their design process using our tools, so they are starting the data entry process which will pay off for them in less work building displays.

¹⁴ Seems like for any given module, a few standard choices would be available: single value (PV) w/ faceplate button, some sort of a graphical SP vs PV deviation element, a 1/4 size faceplate (2 digital values w/ top alarm), a 1/2 size faceplate (3-4 digital values, w/ mode and top 2 alarms) etc...

parameters of the process module.

- 21- The HMI display rendering software has good scaling/zooming/panning features¹⁵, so that displays that aren't perfectly designed to match the operator display framework "view area" still work well.
- 22- Make sure the HMI display rendering software accommodates identifiable "views" on "display-drawings" that are much bigger than the available framework view area. With scaling/zooming/panning features to create a view dynamically, it suggests that these "dynamic views" should be preserved in the sense of the "last display stack", etc.
- 23- Special HMI display rendering algorithms that can help make too dense displays (too zoomed out) workable:
 - An adjustable "details level" which will prevent display elements that have shrunk below a certain size from being reduced below a minimum sized "target" or completely omitted from the display.
 - Making sure that most graphical elements representing plant equipment (vessels, pumps, etc. especially pipes) don't disappear altogether.
 - A means to identify certain "important" data elements as "shrink resistant" (they retain a readable size while other display elements shrink proportionately when zoomed out).
 - A means to test the zoom effects at display engineering time (no live data available).

Net result: a highly zoomed out display still shows the major graphical elements (piping, vessels, pumps, etc.) sufficient to allow the user to choose where to zoom in, and optionally a few of the most important data values.

24- "Display styles" that can be applied to display-drawings that determine things like:

- color palette
- background (tiled) bitmap
- default "details level" setting
- pipe, pump, vessel style (e.g. line drawing, filled, 3d with light source, etc.)
- transmitter style
- valve style
- FF device style
- HART device style
- Interaction preferences (zoom gesture and behaviors, panning gesture, hover times, etc.)
- :

There should be the following levels of display styles that determine the final rendered display:

- A system display style (determines the common display look, alarm priority colors, etc.) for entire system.
- A rendering workstation (or device) display style, which can override properties in the system display style, to deal with color depth limitations in the rendering device, need for graphically simpler and/or higher contrast displays due to ambient conditions, etc.

2.3.2 Internet-accessible HMI for DeltaV

The primary objectives for an internet accessible HMI for DeltaV are:

¹⁵ Including automatic scaling of font based information.

- Low cost (and low maintenance) accessibility to DeltaV HMI display information from virtually anywhere (i.e. from popular web browser(s) running on diverse platforms, connected via the Internet). Wide accessibility for "casual" users (not intended as the primary operator interface for a DeltaV system).
- Security for the DeltaV system by providing a view-only interface.
- Must be able to use the "primary operator interface" displays/drawings. (In most cases it will not be cost effective to have to (re)engineer a set of display definitions for this purpose.)
- At the same time..."casual user" displays for Internet-connected users are likely to differ in design from the highly interactive "primary operating displays". Suggests that these Internet-connected users would need means (via their Internet connections) to create their own displays; often managed separately from the "primary operating displays"¹⁶.

2.3.2.1 Client System Requirements

To maximize accessibility from diverse platforms, and minimizing the effort to prepare a client system to use the DeltaV HMI, it suggests that the client "system" is nothing (much) more than a web browser that is already likely to be in place.

25- The client system application through which the Internet-accessible HMI for DeltaV is accessed will be a leading web browser application. Compatibility testing will be performed with:

- Microsoft Internet Explorer 6.x (and later)

26- There will be no DeltaV-related software HMI client software CD-ROM (or installation) required. Any DeltaV-related software required on the client system will be distributed over the Internet. Ideally any software needed for the Internet-accessible HMI for DeltaV (e.g. ActiveX controls, .Net assemblies, etc.) will be distributed/installed automatically when first needed on the client system.

2.3.2.2 Internet-accessible HMI Features

- View-only access to the DeltaV system. Cannot manipulate DeltaV parameters or perform any actions that change the DeltaV system or affect other DeltaV users.¹⁷
- Display elements showing information from the DeltaV system will be rendered as closely as possible to that when the same display is used on the primary operator interface HMI application.
- Display elements used to cause changes to the DeltaV system (write parameters) need not be rendered as they will not be usable.
- Display elements used to navigate to other displays will be rendered and when activated, will cause the display to change.
- Alarm filtering

2.3.2.3 Server Administration

The administrator of the server for the Internet-accessible HMI for DeltaV must be able to:

¹⁶ These "casual user" displays still need to be managed by the configuration system. To support this capability users can set up additional folders in the display system to store their displays. These displays will only transfer to workstations where the folders have been assigned.

¹⁷ Marketing is requesting that we replace our current Web-based interface. This interface does not allow user to make any changes. This interface is primarily used to support access for users who are 1) not allowed to interfere with the operation of the plant and 2) users who have full capabilities but would like to have view-only capabilities when they are away from the plant. In some cases the Internet-accessible Process Graphics can also be used to support advanced features such as plant optimization, scheduling, etc where access is required to monitor the operating state of the plant.

- Control who may gain access to this feature (via user name and password authentication)
- May view information on who is "currently" connected
- May control which user (or groups of users) have access to which displays (directories)

2.3.3 Wireless (Handheld) HMI for DeltaV

With greater experience using highly portable computing devices with wireless connections, we should expect to find more and more DeltaV users wishing to (and feeling comfortable with) applying those technologies to the monitoring and operation of their control systems.

Possibilities would seem to include:

- For Tablet PCs:
 - Highly portable access to plant drawings that (when near an access point) have live (view-only) signal values and alarm information.
 - A secondary operator interface conveniently available on the same device serving other functions (outside the scope of the DeltaV system) e.g. site inspection logs, machine maintenance administration systems, manufacturing execution systems, etc.
 - A primary operator interface in a (occasionally manned) remote "operators shack".
- For Personal Digital Assistant (PDA) devices:
 - Always accessible (view-only) monitoring of key system "health" or "production target" measures by plant supervisory and engineering staff.
 - Very task-specific control system interfaces for staff working in the plant¹⁸:
 - Accessing a series of I/O values measured by the control system and verifying them with manual observations of the corresponding equipment.
 - While performing minor maintenance on a reactor that is still in use, having a specialized "gauge" to keep an eye on several key measurements (temperatures, agitator speeds, outlet flow rates); perhaps even to warn (annunciate) when a limit is crossed.
 - Using the control buttons in a faceplate to manipulate a control element (valve, motor, etc.) while observing the equipment to diagnose problems.
 - Recording of "operator comments" while performing manual operations associated with a specific unit or batch.
- For SmartPhone devices:
 - Widely accessible (view-only) monitoring of key system "health" or "production target" measures by plant supervisory and engineering staff.
 - Pager-style notification of significant system events requiring attention.
 -

2.3.3.1 Client System Requirements

As the capabilities (and size) of the wireless devices decrease, it would seem that there are more benefits to extremely "thin client" applications in those devices; to minimize the consumption of resources in the device, and the effort to deploy the client applications to the device.

On the PDA platform, use of Terminal Services technology (and the already existing Terminal Service Client) as the client-side application for DeltaV HMI would largely remove the client application deployment issue (and correspondingly reduce the need to develop (the skills in the organization to produce) applications for the PDA operating system (e.g. Pocket PC).

¹⁸ Writing from PDAs, Tablets, and Smart Phones is a required capability. To support these requirements the security infrastructure in Starburn needs additional capabilities along the lines of Profiles in Windows XP and Longhorn.

2.3.3.2 Tablet PC based HMI Features

While the viewing area available in Tablet PCs is comparable to those in traditional desktop workstations used as "operator stations", there are specialized features required in a HMI that works well on these devices:

- 27- These devices support changing the display orientation from "landscape" to "portrait", so aspect ratios for display rendering is likely to be quite different (at times) than the "native" one chosen when the display was engineered. Would seem to require:
 - Particularly good zoom/pan features in the display rendering application (e.g. moving the stylus toward the edge of a display following a pipe causes the display to pan so user can trace where the pipe leads. A magnifier (something like the Windows magnifier) that follows the stylus showing that vicinity of the display in more detail; enough to read data links)
 - Support for selecting different display frameworks depending on current aspect ratio of the HMI managed display area.
- 28- Keyboard-less operation is probably the normal use mode. Suggests requirements for specialized data entry controls which have easy to use numeric keypads (or jog buttons, stylus, etc.) and/or specially designed confirmation actions to reduce the possibility of unintended data entry. More use of selection lists (e.g. to pick from operator selectable enum state names.).

2.3.3.3 PDA based HMI Features

The limited screen real-estate on PDAs means that many applications built for desktop systems are extremely difficult to use if directly transported (or used as a terminal client) on a PDA. This suggests some research into how to effectively design HMI applications (including ones that involve critical interactions with a control system) for that platform.

- 29- More reliance on faceplate variations (e.g. 1/4 size) optimized to fit the PDA form factor.
- 30- Ways of creating task-oriented displays dynamically (e.g. selecting parameters to add to a "Watch Window"). A means to define local "annunciate rules" to parameters being watched.
- 31- Being able to dynamically change display styles (skins) (e.g. to get a high contrast style in difficult lighting conditions.)
- 32- Alarm banners and alarm summaries optimized for the PDA display area.
- 33-

2.3.4 Display Configuration

- 34- Store displays as XML scripts. This includes the following:
 - a. DB adds support for a display object. Includes export/import support and making displays versionable items.¹⁹
 - b. UI adds download management (not downloading if a display is checked out). We would also check to make sure the picture is checked in before doing a download, unless the user has checked the "download even if checked out" box.
- 35- Displays are shown in DeltaV Explorer in Display Folders (and sub-folders) under System Configuration.
- 36- Displays, either class-based or not, can be assigned to both class-based and non-class-based modules. Currently Modules contain the following information:
 - a. PRIMARY_CONTROL_DISPLAY

¹⁹ Versioning is on the display level. This is already in place in DeltaV 7.3. This feature is for users who have enabled VCAT.

- b. INSTRUMENT_AREA_DISPLAY
- c. DETAIL_DISPLAY

These properties will be left as is.

In addition, a list of (Display Type, Display/Display Class) pairs will be added. This way we can generalize the application of e.g. thumbnails, outlines, detail displays, faceplates, maintenance displays, or whatever types make sense. This potentially facilitates the runtime selection of sub-display based on equipment mentioned later.

37- A formal interface will be used to handle check in/checkout management tied to displays.

38- A DB-ID will be assigned to displays. This allows us support VCAT behavior.

39- Display File Names must be unique across the system²⁰.

40- The display editor can be launched from DeltaV Explorer.

41- Manage download indicators on the Operator Subsystem and the Displays Folders. This includes the following:

- a. The Operator Subsystem on Workstations will be used to provide download indication (i.e. displays need to be downloaded).
- b. The Display Folder under System Configuration (similar to Recipes) will also be used to provide download indication.
- c. The download indicator for non-class-based displays is set on all nodes where the display is assigned. This means that when a non-class-based display is updated
 - The Operator Subsystem download indicator trips on relevant workstations.
 - The download indicator on the Display, Display Category, and Displays Folder trips under System Configuration.
- d. The download indicator for class-based displays is set on a system-wide basis. This means that when a class-based display is updated
 - The Operator Subsystem download indicator trips on all workstations where control modules reference the class-based display.
 - The download indicator on the Display, Display Category, and Display Folder trips under System.
- e. Update download status will regenerate the download indicator on displays by computing checksums on the displays.

42- Download options:

Non-class-based displays are downloaded to all workstations. This is the same behavior that current systems have. Users should be allowed to control exactly which graphics get downloaded to a workstation.

- a. Class-based displays follow the Area assignments for Units, Equipment Modules, and Modules on workstations.

The context menu of a display in the library includes Download and the user can elect to download the display to all workstations to which it applies.

- b. Context menu of Operator Subsystem includes Download All displays, Download New and Changed Displays.
- c. Allow users to download all displays from the Displays Folder. This is consistent with a total system download.

²⁰ Since there are 1000's of displays in a typical system it makes sense to provide a file structure in the runtime. The file structure used to store displays in the runtime is not yet known.

- d. Allow users to download one Display Folder at a time. There is no display like this today. This is consistent with Recipe Procedure/Unit Procedure/Operation downloads. Note all nodes where the displays are assigned will be downloaded.
- 43- To support downloads a well defined download directory will be available on all workstations connected to the ACN.
- 44- Displays will exist once per database, not once per system as they are now. If the active database is changed the whole system needs downloading anyway so that should not be a problem.
- 45- Foreign documents can be associated with both Class-based and non-Class-based displays. This allows the Process Engineers to provide operating instructions associated with Displays that Operators could use to assist in using the item the display is associated with.
- 46- The display runtime will load the display class and lookup the correct bindings for the item being instantiated in the runtime. When invoked the display would be associated with a particular instance of the associated class based item in the equipment hierarchy. Most bindings would be relative to this but there may also be some absolute bindings.
- 47- Changes to DeltaV names will automatically be updated in the Display Binding Table and reflected in the download indicators.
- 48- A Utility will be developed to covert non-Class-Based displays to Display Classes with Display Binding Tables.
- 49- The display runtime will load the Display Class referenced in the DeltaV Modules and lookup the correct bindings in the Display Binding Table.
- 50- The Display Editor will allow for the definition of a Display Class as well as a thumb nail view of the Display Class. This will allow us to create a layered view (e.g. a Unit with Equipment – the equipment is shown as a thumb nail – when selected the equipment view opens up to full view).
- 51- New maintenance displays will be created. These Class-based Displays will be associated with Nodes, Cards, Devices, etc.
- 52- New conditioning monitoring displays will be created. These Class-based Displays will be associated with Condition Monitoring items such as Loop monitoring, Device Alerts, etc.

2.3.5 Report Generation²¹

The Starburn architecture must support a full featured report generator. Key requirements for the Process Graphics environment include:

The foundation for improving plant operating operations and efficiency is the ability to accurately track the material and energy used in production, the final product produced and the efficiency of production. Such information is most useful when it is possible to view or trend historical values and to display total or averages over a period of time e.g. shift, day, month. Support for this type of information at the Operator Display is one aspect – but what happens if users want this information by shift, for last week, last month? This suggests the following:

- 53- The Report Generator should be able to ask the Process Graphics system to load a display, pull data from the RT, Database, OPC, and other sources, and save itself into an XAMAL format suitable for embedding into a Runtime Report.²²
- 54- Other elements, such as trends and graphs, can also be included into the Reports.
- 55- Since the users can specify time, the Process Graphics engine will be able to locate values and pull data values from the historian.
- 56- Users can specify a specific time, e.g. hour, shift, day or monthly total or average.

²¹ At this point we are thinking that we will use MS Word document format. MS Word is familiar with most users and it is a rich format.

²² MS Word will support XAML format in the Longhorn time-frame.

- 57- Users can specify a relative time, e.g. an hour, shift, day or month total or average relative to the current hour, shift, day, or month.

For example, by configuring a specific hour, then the value shown in the display will always correspond to that specific hour i.e. change once per day. However, by selecting relative data, then the latest total or average for that timeframe will be displayed. For example, if the graphic element is configured as a relative hourly reference (with zero hours offset), then the last hour total or average would always be displayed i.e. value would change at the end of each hour.

- 58- The display system can request data from a wide range of services – RT, Historical, OPC, Events, as well as other sources.
- 59- The following will need to be configured for the total/average graphic element:
- Path to the function or process block parameter.
 - Total or Average Value Selection.
 - Relative or absolute value desired.
 - Relative offset to current (configured for relative values).
 - Scaling factor (in case the units of the measurement are not in minutes e.g. GAL/HR).

2.3.6 Conversion of Process Graphics to Process Modules

A Process Module may be generated from a Process Graphics. In this case the functionality available to the Process Module is determined by the Process Graphic elements. When the user configures their Process Graphic Display they have the ability to include additional information such as Mass or Energy Streams. This streams information is also provided to the Process Module.

Since Process Modules are a new type of DeltaV Modules, it is possible for them to reference, and be referenced by, DeltaV parameters, control strategies, displays, etc as illustrated below. Also, using this capability, it is possible for a process module to be created in DeltaV Simulation Studio independent of the Process Graphic Display.

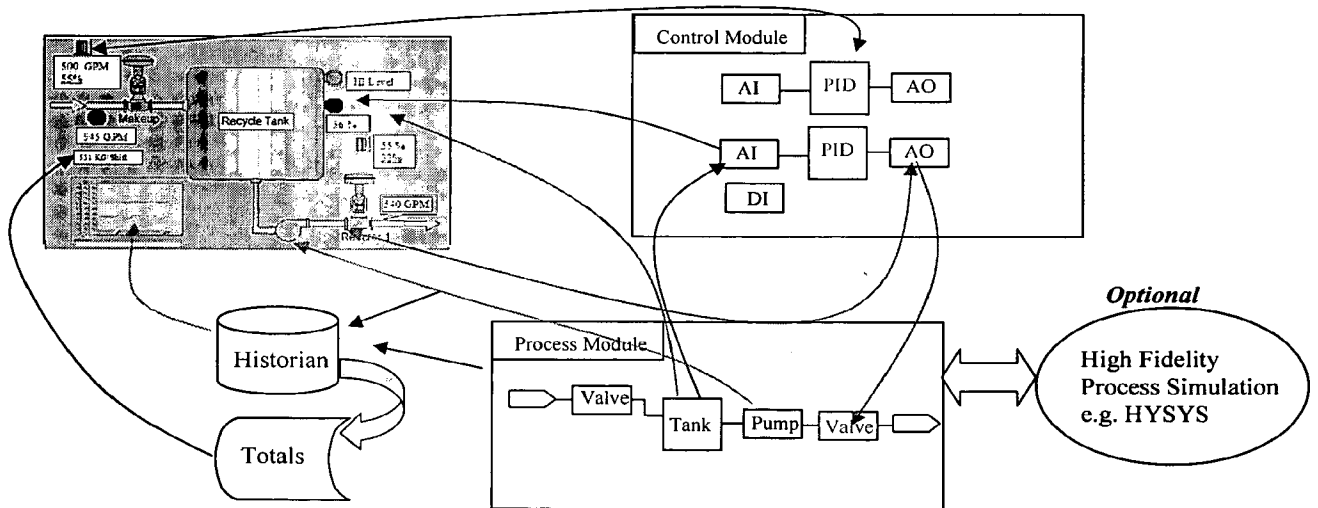


Figure 6. Interaction of Process Graphics with Process Modules

It is also possible to generate Expert and MSPC algorithms from Process Graphics – refer to the Process Modules section for details on this.

Key requirements in this scenario include:

- 60- From the Process Modules Editor, be able to load a Process Graphic Display.
- 61- Use the Process Graphic Display elements, together with it's Schema, to generate Process Module Elements.
- 62- Preserve references that were configured into the Process Graphics as part of the Process Modules interface.
- 63- Preserve type information, over-rides, etc from the Process Graphic Display.

2.3.7 Conversion of INtools to Process Graphics

The Starburn vision document specified that we should be able to convert INtools P&ID's to DeltaV Process Graphics. Key requirements in this scenario include:

- 64- Convert the P&ID graphics to Starburn Graphics.
- 65- Preserve device and equipment information from the P&ID.
- 66- Preserve information on how process elements are connected.

We met with the Integraph team in Huntsville last year. The support we are looking for is available through their engineering framework.

2.3.8 Browse

The Starburn Process Graphics environment maintains it's own internal schema. It uses it's schema to manage itself. In order for the DeltaV Process Graphics Editor, DeltaV Explorer, etc to browse displays the Process Graphics infrastructure will need to support browsing. Key requirements in this scenario include:

- 67- Implement a Browse Interface specified as part of the Browse Framework.
- 68- Allow Process Graphics to reference parameters internal to the display.
- 69- Expose parameters at lower layers in a particular display as named parameters at higher levels in the interface.

2.4 Display Technology

2.4.1 Rich Display Client

Rich display clients will continue to be used as they are today, i.e. for dedicated operator interfaces. The same download scripts sent to rich clients also work with the web-based UI's.

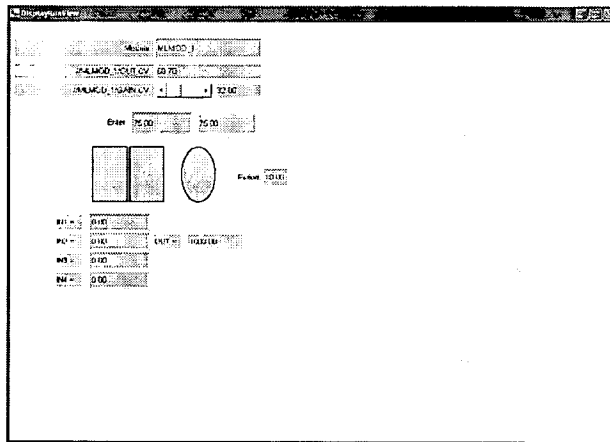


Figure 7. Rich Client

2.4.2 Web-based Client

The Web-based clients read XML files. Dynamic behavior is provided through a combination of predetermined functions and C# Scripts. Data is transferred through to the displays over Web Services.

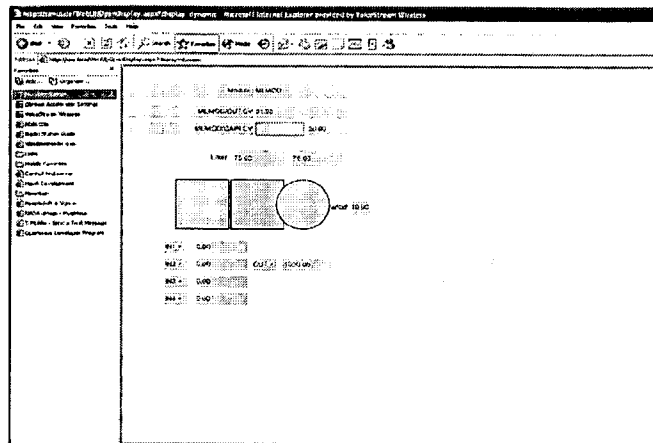


Figure 8. Web Client

2.4.3 Smart Handhelds

Smart handhelds can be also be supported.

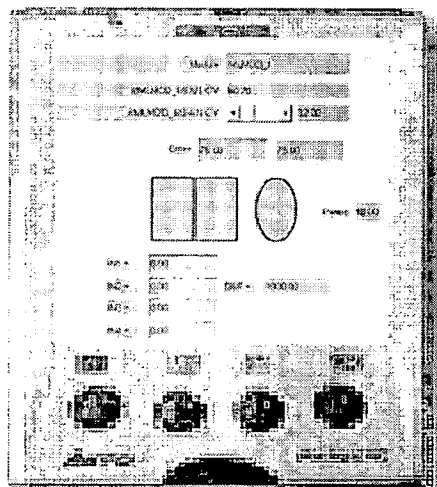


Figure 9. Smart Handhelds

2.4.4 Smart Phones

Smart phone investment is being driven by the consumer industry. New gaming technology is further driving the capabilities of these devices. Because of the extremely limited screen size and keyboard it will be necessary to provide specialized displays (postage stamps). The same configuration environment could easily generate these displays by providing hints on what the target display device will be (i.e. select canvas size).

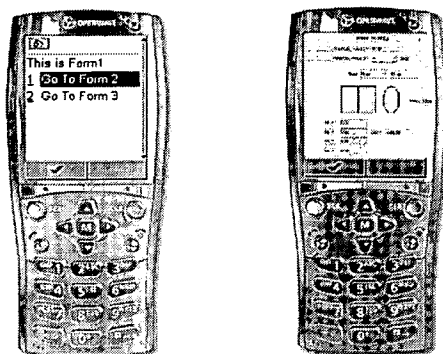


Figure 10. Smart Phones

2.5 Process Graphics Architecture

2.5.1 Process Graphics

Underlying Process Graphics is Microsoft's XAML implementation; XAML together with Indigo and WinFS are the key elements of the Longhorn operating system release. The XAML format specifies the properties of drawing elements for vector graphics commands. A XAML script can include static data as well as dynamic data. The dynamic elements and attributes can be associated with real-time values, historical values, physical properties such as mass-flow and composition, etc. In this sense XAML is data driven.

XAML vector images are stored as text-based commands. Since it is vector based, it is possible to have the same high quality wherever a display is rendered - whether on smart phones, handhelds, or high-end monitors.

Since XAML is entirely text based it is convenient for DeltaV to store and emit as a download scripts. Using the same techniques used today for control configuration, it is possible to generate download status indicators, be managed as part of configurations, and participate in version control and upgrades. Process Graphics are downloaded in much the same way that IO and Control, System Setup data, Batch, Advanced Control, and SIS configuration scripts are downloaded today. Since Process Graphics are stored as text, it will also be convenient for users to search for text within images.

At its core XAML supports vector graphic shapes consisting of straight lines and curves, images, and text. It supports the basic graphical shapes including rectangles, circles, ellipses, and polygons. It also gives developers the ability to control and implement animation techniques that can be as complicated as generating three-dimensional morphing effects to simple two-dimensional images. The graphical information is stored in a sequence of commands that draw various vector graphic shapes. This information can be converted by various methods to display an application-specific graphics image. Scalability is achieved without jagged edges because redrawing instructions are sent to the rendering program, rather than sent as pixel values in a bitmap. Rendering engines are defined for each target environment.

Every XAML element and every attribute of an element in a Process Graphic Display can be animated. It supports bitmap-style filter effects for creating high-impact graphics. XAML's filter effects give developers the ability to add effects directly to shapes and text in a manner similar to those achieved using professional imaging tools. XAML has the support for blending, tiling, transforming or rotating elements, morphing, offset, merge, and special lighting effects. Filter effects can be applied separately or in any combination on a vector image. Special effects such as mouse over features can be assigned to any XAML graphical object²³.

²³ We need to determine how much of this capability to expose in the first release.

XAML produces small file sizes that allow for fast downloading. Additionally, since DeltaV will include Groups, Composites, and Templates it will be easy to build up a large library of pre-built elements that users can use and add to.

XAML can be drawn using data at run time that is derived from a number of different sources in a distributed computing environment. The data can come from DeltaV Runtime, DeltaV Historian, OPC, an XML file, or many other sources. To accomplish this an abstraction or a logic layer is created to establish the relationship of the data to the XAML graphic. For instance, suppose XAML describes an image that represents data in terms of pixels (e.g. filling a bar). The real-time data may be stored in units of degrees Fahrenheit. It would be necessary to establish some type of logical relationship between the image and the data that will be displayed in that image so the end user can view something in a format that is easily understood and not subject to misinterpretation. Build-in support from DeltaV will provide the scaling before the data value is rendered.

The overall Process Graphics functionality is realized by two major applications. The configuration environment consists of Smart Process Graphics Studio. Smart Process Graphics Studio includes several tools that are used for creating graphics and graphic components and binding graphics to data sources. The second application is the Smart Process Graphics Server. The server provides back-end support for executing Smart Process Graphic Displays. These will be described in the following sections.

2.5.2 Process Graphics Elements and Operations

Process Graphics Elements are the basic infrastructure for Process Displays. Primitive Process Elements include the following:

- 1- Shape Elements
 - a. Rectangle
 - b. Circle
 - c. Ellipse
 - d. Line
 - e. Polygons
 - f. Path
- 2- Container Elements
 - a. Group
 - b. Symbol
 - c. XAML script
- 3- Text Elements
 - a. Text
 - b. Text path
- 4- Reference Elements
 - a. Image
 - b. HTTP Link
- 5- UI Elements
 - a. Button
 - b. Check-box
 - c. Slider
 - d. etc
- 6- Other Elements

- a. C# Script
- b.

For each Element or Element Group the user can define attributes. Example attributes include:

- 1- Tool Tip
- 2- Attach a Context Menu (could launch help from here)
- 3- Define Drag/Drop behavior (e.g. drop a tag on a Plot)
- 4- Zoom and Span

Users can associate behaviors with Elements and Grouped Elements. Behaviors include:

- 1- Action, Alert based on a IO Mapping
- 2- Conditions such (if-then-else, switch)
- 3- Scripts

In the Process Graphics Editor Users can perform the following actions on Elements and Groups of Elements:

- 1- Create, Copy/Paste, Move, Replace, Resize, etc
- 2- Convert to Dynamics – any static element, or grouping of elements, can be converted to a Dynamic element. For example, a user may define a conveyor group, convert the group to a Dynamic Group, expose parameters related to the animation of the Conveyor, and add script to define the movement of the Conveyor.
- 3- Add a Listener (register for an event and take action when the event arrives – e.g. press a button)
- 4- LoadURL
- 5- Load XAML
- 6- Print Element

Actions that the editor can perform on Elements include:

- 1- Cut/Copy/Paste/Delete Nodes and Elements
- 2- Move to Back/Front
- 3- Move One Forward/Backward
- 4- etc

Creating Groups

- 1- Create Group
- 2- Define which elements are to be made Dynamic

- 3- Define functionality for each of the Dynamic Elements
- 4- Expose attributes that need to be bound to Inputs and Outputs

Using Groups

- 1- Drag Groups / Composites from Pallet
- 2- Attributes that can be bound show up in the parameters view
- 3-

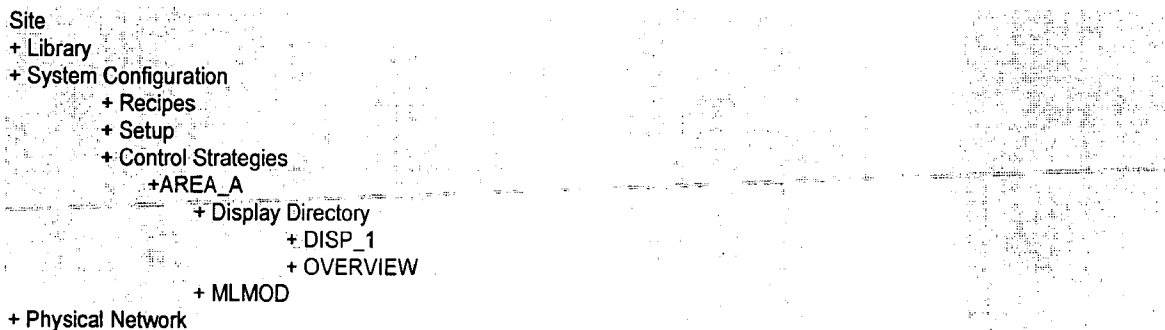
2.5.3 Process Graphics Configuration

A new engineering environment will be provided for the creation of graphic displays. DeltaV Explorer will be modified to allow a user to easily find the displays that have been created. Displays may be viewed by process area. Using a new display editor, it will be possible to quickly create graphic displays. Standard 3-D elements²⁴ may be used to create displays that dynamically show measurements, actuators and process equipment simulation. Also, static elements and dynamos associated with control and calculations will be supported. The user may define display layers to address the interface requirements of different users of the graphic displays. In addition, the capability will be provided in the display editor to create new smart 3D elements, dynamos and static elements. The display editor may be used to automatically generate process simulation based on the smart 3-D elements and their connections defined in a display.

2.5.3.1 Explorer View

The DeltaV configuration for a large process plant will contain numerous process graphic displays. In most cases, the operator for a given process area will need access to the displays associated with their area of responsibility. Through these displays, they may make changes to their process area. Also, an operator may need to view displays associated with upstream and downstream process areas that impact their process area. Other DeltaV stations in the control system may require access to all displays available to an operator plus additional displays for plant engineering, maintenance and operation supervision.

Thus, to make it easy to engineer and maintain DeltaV graphic displays, displays will be assigned to the Operator Subsystem on workstations. Downloads will then work off of the Operator Subsystem. For example, the displays defined for AREA_A will be shown in the hierarchy as illustrated below.



²⁴ 2D objects with 3D rendering.

- + Control Network
 - + USAUST-DELLBERT
 - + Assigned Modules
 - + MLMOD
 - + Operator <<assign displays by dragging areas to subsystem >>
 - + DISP_1
 - + OVERVIEW

When a display is selected, then by right clicking a display name, the option will be provided to launch the Graphics Editor to view or modify the display.

Also, when a display is selected the contents view in explorer will show the top level objects in the display. To support this the Process Graphics subsystem will need to be invoked to extract the top level elements from the stored displays.

The status of each display (downloaded, not downloaded, unknown) will be indicated by the icon provided for each display name in this folder. Also, the icon will be used to indicate if the display is to be locked in memory or whether it can be maintained in virtual memory on the disk.

2.5.3.2 Process Graphics Editor

The Graphics Editor is used to create new displays, composites, templates, dynamos, etc for plant operations, engineering and maintenance. This editor may be launched using the Start/DeltaV/Engineering/Process Graphics Editor selection. When the display editor opens, the user will be presented with a work area, toolbar, and multiple palettes of graphics elements. The operator display layer will be selected by default. To configure the operation view, Graphic elements may be dragged from the palette into the work area. The user connects the elements together by holding down the left mouse button over a connection point and moving the cursor to a destination connector (similar to what they do with Control Studio). The type of connection (Pipe, Conveyor or Duct) will automatically be defined by the definition of the starting connector. When a connection supports multiple types of connections, then the default connection type, defined through the toolbar selection, will be used. When a connection is being created, an elbow/bend will be automatically provided each time the mouse button is released before reaching a defined connector. Individual graphic element is selected by clicking on or near its graphic representation. When a graphic element is selected, then the configurable parameters associated with the graphic element will be displayed in the lower left pane, as illustrated below.

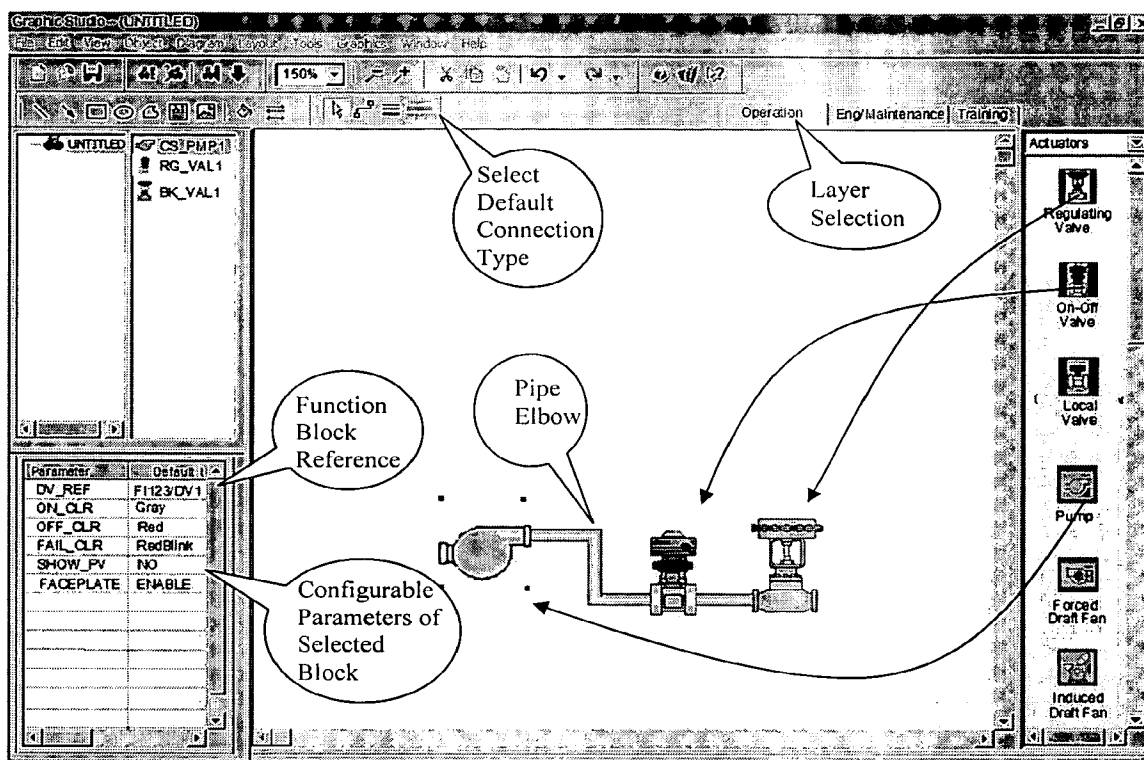


Figure 11. Process Graphics Editor

Actuator and transmitter elements will have one or more function block references. By double clicking on these reference parameters, a dialog will be provided that allows the user to define the path to the desired 'control modules / block / parameter'.

Common processing elements such as tanks, mixers, heat exchanges may be added to a graphic using the Process Equipment Palette. When such elements are added to the display, then they may be stretched to a size that is appropriate for the display. Also, by selecting an element, the number of input and output streams and their position may be modified through the associated parameters shown in the left pane, as illustrated below.

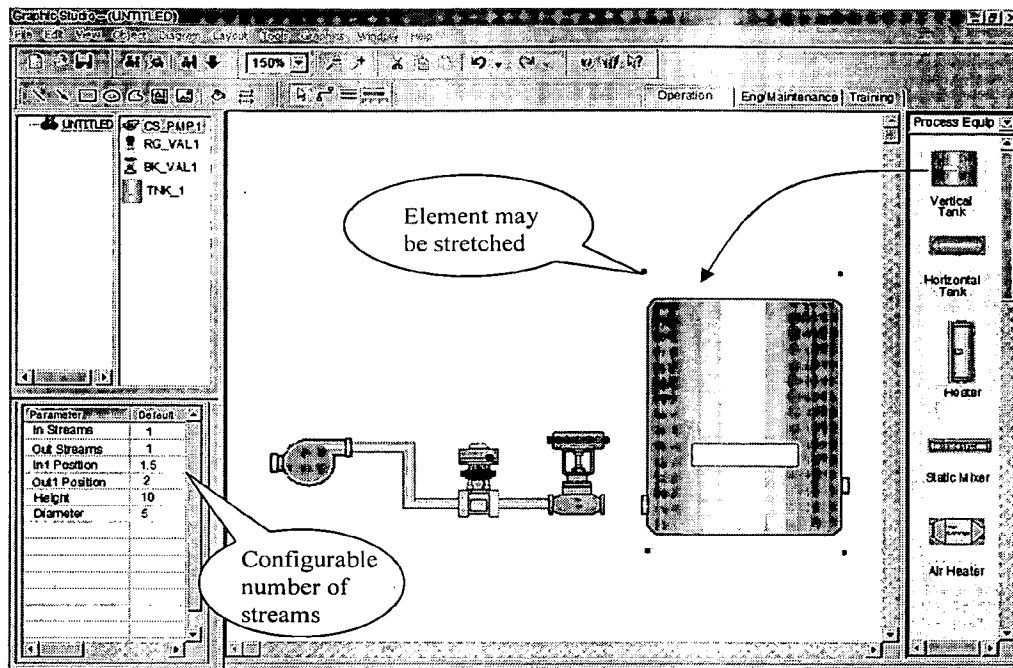


Figure 12. Using Process Equipment Palette

A Custom Equip palette of graphic components will be supplied for less common processing equipment. Also, the capability will be provided to allow the user to modify or define new Process Equipment components.

Analog and discrete measurements that are included in the plant may be identified by dragging a transmitter or switch from the Measurement palette and positioning it over the associated stream (pipe, conveyor or duct) or a piece of equipment, as illustrated below.

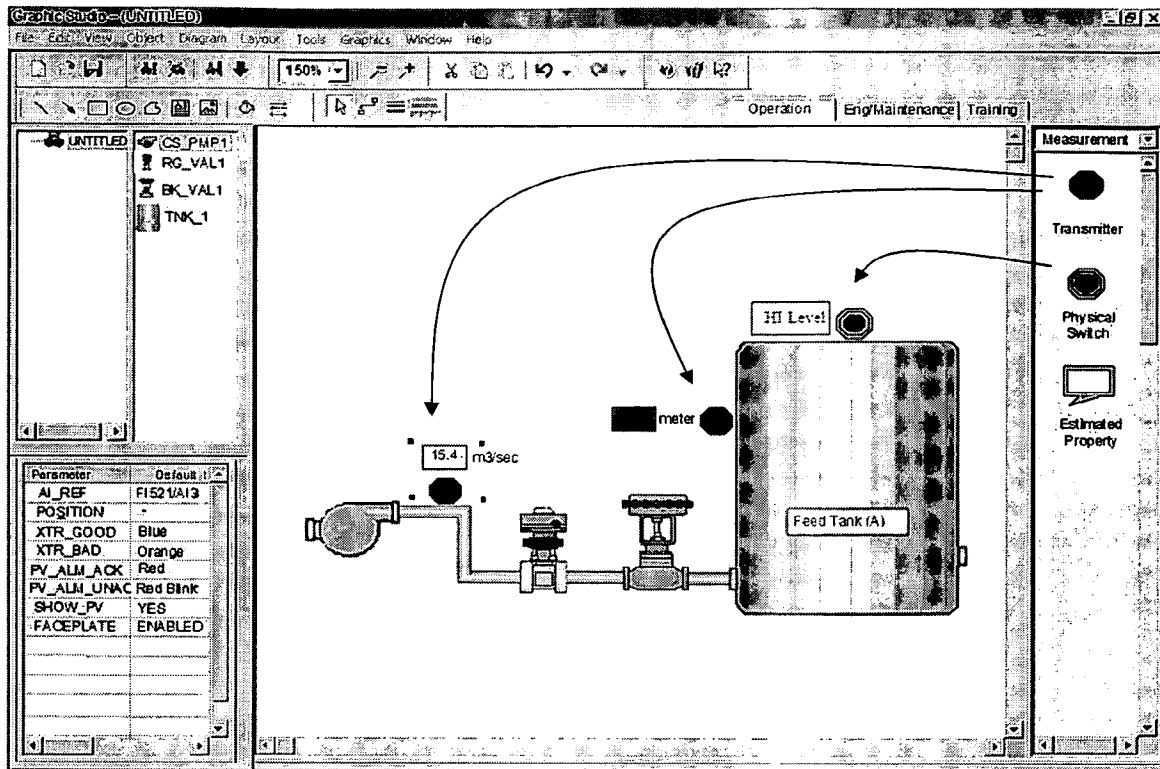


Figure 13. Using Measurement Palette

Through the configuration parameters of measurement blocks, the user may select the colors applied to the device and the PV value background to indicate the state of a device (good or bad) and process alarms (in alarm an acknowledged or unacknowledged).

A Special palette will be provided to show starting and ending points of a stream contained on a graphic and to allow a graphic display to be represented within a display. Internal stream references allow the source of the stream to be named. External stream references include parameters that may be used to identify the associated stream on another display. The thing that must be configured for an external output reference is the name. External input references configuration identified the Display and output reference associated with the input. One external reference parameter may be configured to reference an output parameter. The name of the display and external output reference is shown on the associated external input reference, as illustrated below.

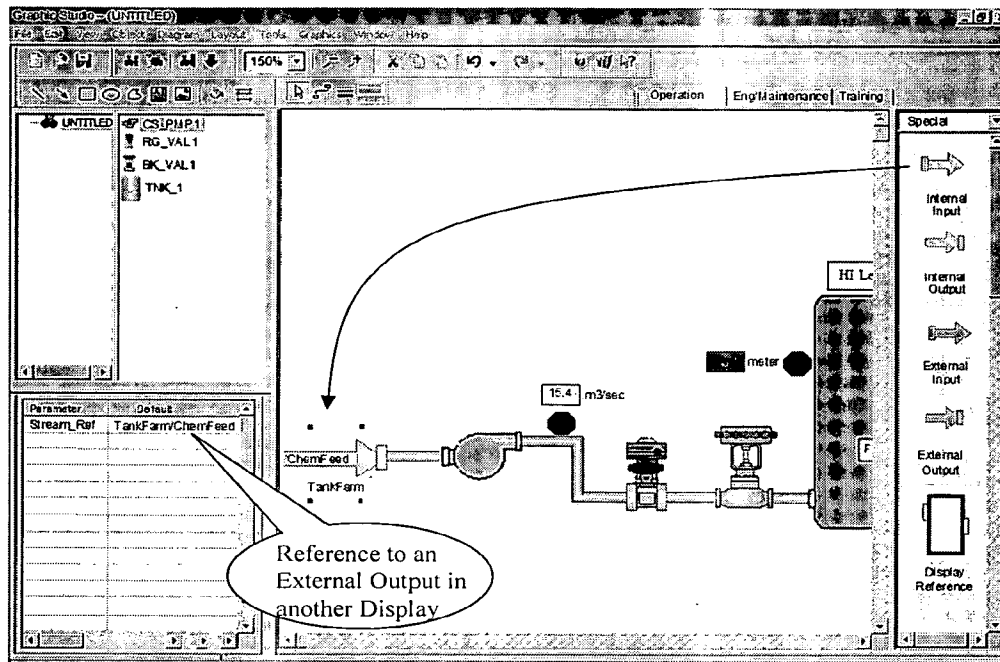


Figure 14. Special Pallet

The details associated with a unit within a process may be provided in a dedicated display. This unit may be included in another display by adding a Display reference. The name of the referenced display will be shown within the Display reference, as illustrated below

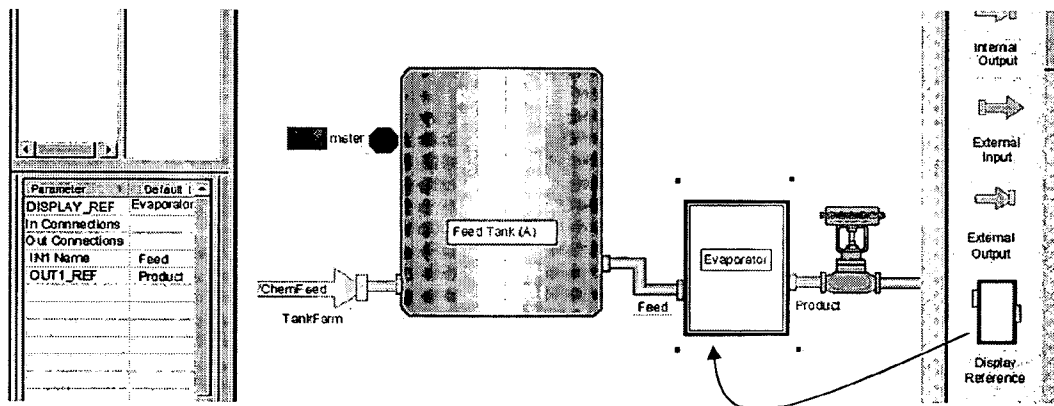


Figure 15. Display Reference

By selecting the Eng/Maintenance layer, the user may define components that will be shown on the display if the person who logs into the node has engineering or maintenance privilege. For example, if on-line simulation of the process is implemented, then estimated stream properties or properties of a unit, such as calculated operating efficiency may be exposed using the Estimated Property element provided in the Measurement palette. For example, the calculated pump discharge pressure could be shown using the Estimated Property element, as illustrated below.

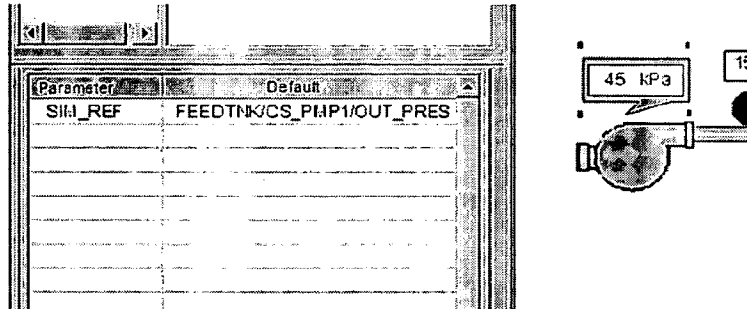


Figure 16. Engineering and Maintenance Views

2.5.3.3 Tree View

The Tree View in the Process Graphics Editor is used to show the Elements, Groups, Composites, and other top level items (except parameters) in a Process Graphic Display. The user can expand elements in the Hierarchy in a manner similar to what they do with Control Modules.

2.5.3.4 Parameter View

The Parameter View in the Process Graphics Editor is used to show all of the parameters for the selected item in the contents or Tree View. Parameters that are to be bound for the purpose of Animation, Events, etc are included in the Parameter View.

2.5.3.5 Dynamic Behavior View

Users can add dynamic behavior to any display element.

Need to add a few screens here showing how dynamic behavior is added.

2.5.4 Process Graphics Display Format

A Process Graphics display definition is contained in an MSAvalon XAML file. This file represents an operator interface screen, and is loaded and processed by the edit and runtime applications.

The file structure can be divided into two main sections: Display and Dynamics. These are contained in a panel-type MSAvalon UI element (see {Panel} tag below) like Canvas, DockPanel, etc.

```
<{Panel}
  xmlns="http://schemas.microsoft.com/2003/xaml"
  xmlns:def="Definition"
>
  <{Panel}.Resources>
    <!-- Begin Dynamics section -->
    ...
    <!-- End Dynamics section -->
  </{Panel}.Resources>

  <!-- Begin Display section -->
  ...
  <!-- End Display section -->
</{Panel}>
```

Figure 17 – Process Graphics Display XAML File Structure

2.5.4.1 Display Section

The Display section defines the Process Graphics objects or figures that can be seen by the user. This consists of MSAvalon UI element tags. For more information, please refer to Microsoft Longhorn documents.

```
<Canvas
  xmlns="http://schemas.microsoft.com/2003/xaml"
  xmlns:def="Definition"
>
  <Canvas.Resources>
    <!-- Begin Dynamics section -->
    ...
    <!-- End Dynamics section -->
  </Canvas.Resources>

  <!-- Begin Display section -->
  <Canvas ID="MyPic" Canvas.Top="10" Canvas.Left="10">
    <Ellipse ID="Ellipse1" />
    <Line ID="Line1" />
    ...
    <Button ID="Button1">
    ...
    </Button>
  </Canvas>
  <!-- End Display section -->
</Canvas>
```

Figure 18 – Display Section

2.5.4.2 Dynamics Section

The Dynamics section defines the dynamic runtime behaviors of Process Graphics objects in a display. In the current prototype implementation, there are three parts that make up the dynamics of a display:

- Dynamics Definition
- Dynamics Instances
- Event Dynamics

Each of these is defined as an **XmlDataSource** object in the **Resources** tag of the outermost panel. The **XmlDataSource** is used since it can contain custom XML data structures within an XAML file.

```
<Canvas
  xmlns="http://schemas.microsoft.com/2003/xaml"
  xmlns:def="Definition"
>
  <Canvas.Resources>
    <!-- Begin Dynamics Definitions -->
    <XmlDataSource def:Name="DynamicsDef" XPath="Dynamics">
      <Dynamics xmlns="">
        ...
      </Dynamics>
    </XmlDataSource>
    <!-- End Dynamics Definitions -->
    <!-- Begin Dynamics Instances -->
    <XmlDataSource def:Name="DYN_INST_{NAME1}" XPath="DynamicsInstance">
      <DynamicsInstance xmlns="" ...>
        ...
      </DynamicsInstance>
    </XmlDataSource>
    <XmlDataSource def:Name="DYN_INST_{NAME2}" XPath="DynamicsInstance">
      <DynamicsInstance xmlns="" ...>
        ...
      </DynamicsInstance>
    </XmlDataSource>
    <!-- End Dynamics Instances -->
    <!-- Begin Event Dynamics -->
    <XmlDataSource def:Name="EVENT_{NAME1}" XPath="EventDynamics">
      <EventDynamics xmlns="" ...>
        ...
      </EventDynamics>
    </XmlDataSource>
    <XmlDataSource def:Name="EVENT_{NAME2}" XPath="EventDynamics">
      <EventDynamics xmlns="" ...>
        ...
      </EventDynamics>
    </XmlDataSource>
    <!-- End Event Dynamics -->
  </Canvas.Resources>
  <!-- Begin Display section -->
  ...
  <!-- End Display section -->
</Canvas>
```

Figure 19 – Dynamics Section

2.5.4.3 Dynamics Definition

The Dynamics Definition contains all the dynamics definitions to be used in executing the runtime behaviors of a Process Graphics display. The Dynamics Definition section is defined using the MSAvalon XmlDataSource object tag with the def:Name "DynamicsDef". Except for the content within the CDATA section, the XmlDataSource tag attributes, values and structure must follow the one shown below:

```
<XmlDataSource def:Name="DynamicsDef" XPath="Dynamics">
  <Dynamics xmlns="">
    <![CDATA[
      function SWITCH(cur, val1, val2, test, testval)
      {
        if (test == testval)
          return cur;

        if (cur == val1)
          return val2;
        else
          return val1;
      }

      function ADD(val1, val2)
      {
        return val1 + val2;
      }

      function ADD_MAX(val1, val2, valmax)
      {
        var sum = val1 + val2;
        if (sum > valmax)
          sum = val1;
        return sum;
      }
    ]]>
  </Dynamics>
</XmlDataSource>
```

Figure 20 – Sample Dynamics Definition

Within the CDATA section is defined all the dynamics execution scripts. Each script function represents a specific dynamics definition to be used within the display. In the example above, the SWITCH() function represents a Switch logic with test condition. The CDATA wrapper is used to tell the consumer of this XAML file that the content contains custom data and should be treated "as is" and not as XML.

2.5.4.4 Dynamics Instances

A Dynamics Instance pertains to a specific application instance of a certain dynamics definition within the display in relation to actual input and output parameters. This means that within a display XAML, there could be zero or more than one dynamics instances specified. Each dynamics instance is defined using the MSAvalon XmlDataSource object tag with the def:Name prefixed with the word "DYN_INST_". The following shows the structure of a dynamics instance:

```
<XmlDataSource "DYN_INST_MYDYNINST" XPath="DynamicsInstance">
  <DynamicsInstance xmlns="" func="MYFUNCTION(INNAME, ..., INNAME_N, 0, ..., 1)">
    <Out name="OUTNAME" source="SOURCE" path="PATH" value="VALUE" />
    <In name="INNAME" source="SOURCE" path="PATH" />
    ...
    <In name="INNAME_N" source="SOURCE" path="PATH" />
  </DynamicsInstance>
</XmlDataSource>
```

Figure 21 – Dynamics Instance Structure

The func attribute is where the function signature of the dynamics definition to be used is specified. A function parameter could be a constant or a reference to an input (In) parameter.

Under the DynamicsInstance XML node, one Out parameter and zero to more than one In parameters can be specified. An Out parameter represents the object or link to which the result of executing the associated dynamics definition is assigned, and In parameter represents the object or link from which a parameter should get input data from. Both In and Out parameters contains the following attributes:

- **name** Name or alias of the parameter. For both In and Out parameters, this is used to replace the references listed in the function signature with the current value of the object or link.
- **source** Source of the parameter. Each source, e.g. "DISPLAY" for display objects "RT" for DeltaV runtime datalinks, "HISTORY" for history objects, etc. must be defined.
- **path** Path of the parameter. If the source is "DISPLAY", path should take the form of "[ui:element_id].[property]". For example, "Line1.X1" pertains to the X1 property of the UI element from the Display section with id "Line1".

The Out parameter has an extra attribute called value, where the initial value of the output parameter is specified. This is required since the XML node attribute is bound dynamically to the UI element associated with the Out parameter. This way, the container may update this node attribute to change an UI element's property without going through a manual search for that element within the graphics object model by name.

2.5.4.5 Event Dynamics

An Event Dynamics pertains to a specific event application instance of a certain dynamics definition within the display in relation to actual input and output parameters. This means that within a display XAML, there could be zero or more than one dynamics instances specified. It is very similar to Dynamics Instances, except that the associated dynamics definition is executed when a particular event is triggered, and the def:Name is prefixed with the word "EVENT_". The following shows the structure of an event dynamics:

```
<XmlDataSource "EVENT_MYEVENT" XPath="DynamicsInstance">
  <EventDynamics xmlns="" func="MYFUNC(INNAME, ..., INNAME N)" event="OBJID.EVENT">
    <Out name="OUTNAME" source="SOURCE" path="PATH" value="VALUE" />
    <In name="INNAME" source="SOURCE" path="PATH" />
    ...
    <In name="INNAME_N" source="SOURCE" path="PATH" />
  </EventDynamics>
</XmlDataSource>
```

Figure 22 – Event Dynamics Structure

The addition to the structure of the event dynamics in comparison to the dynamics instance is the **event** attribute, which specifies the control object and the name of its event to link the dynamics to. In the example above, to link EVENT_MYEVENT to the **Click** event of the **Button** with id="MyButton", assign "MyButton.Click" to the attribute.

2.5.5 Process Graphics Runtime User Interface

The Process Graphics Runtime application is implemented as an MS Avalon Navigation Application that consists of a main window and multiple floating child windows. Each window is capable of processing a Process Graphics Runtime XAML file, displaying the graphics objects, and running the dynamics behavior defined.

The main window is an MS Avalon NavigationWindow consisting of a window container that hosts Process Graphics Runtime display XAMLs.

2.5.5.1 Loading Displays

The first and most basic feature is the loading of display XAML files and the dynamics defined therein. The following diagram shows the object collaboration supporting loading:

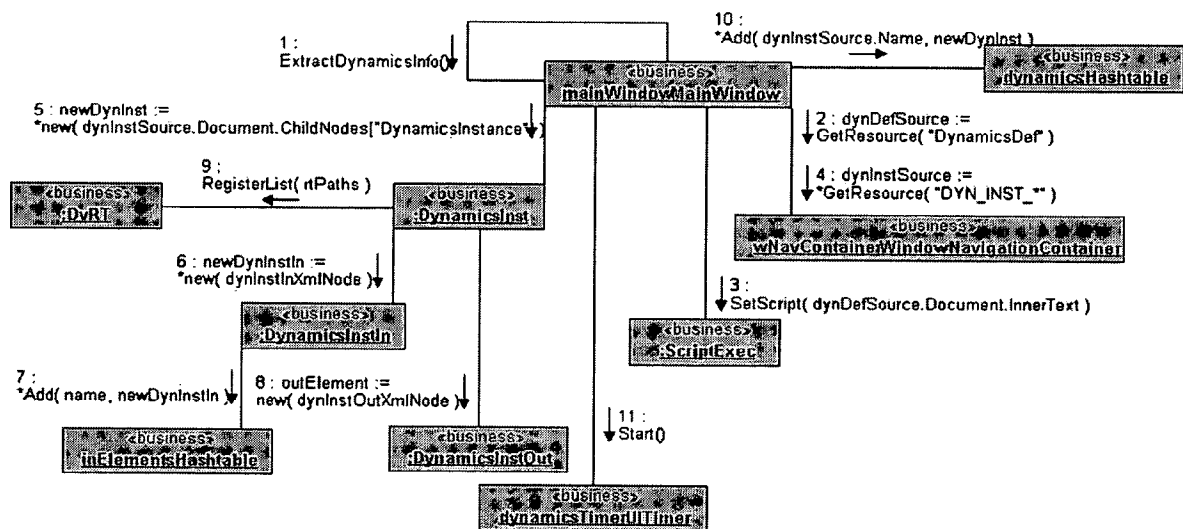


Figure 23 – Object Collaboration Diagram for Loading a Display

Upon loading a display XAML file, the container window processes the dynamics sections and creates the appropriate objects to represent them. **Out** parameters of **Dynamics Instances** are dynamically bound to the specific UI element's properties, while the generic event handler function of the window is set to listen to the events of the control objects specified in **Event Dynamics**.

The created dynamics objects are then saved into a hash table, which will be processed one by one within a dynamics timer thread. The script in the dynamics definition section is set in a script executor (ScriptExec) for later use. Once everything's processed, the dynamics timer is started. Currently one dynamics timer is supported within the container window.

2.5.5.2 Dynamics Processing

The container window utilizes a timer object to process the dynamics specified in the **Dynamics Instance** sections at a given time interval (default 100ms). During each interval, each dynamics instance is processed by creating the right function signature with the appropriate parameter values from the specified parameter sources. The resulting signature is then sent to the script executor (ScriptExec) for execution. The ScriptExec dynamically compiles all the scripts if it hasn't done so before actually executing the scripts. The result of the execution is then set to the appropriate **Out** parameter.

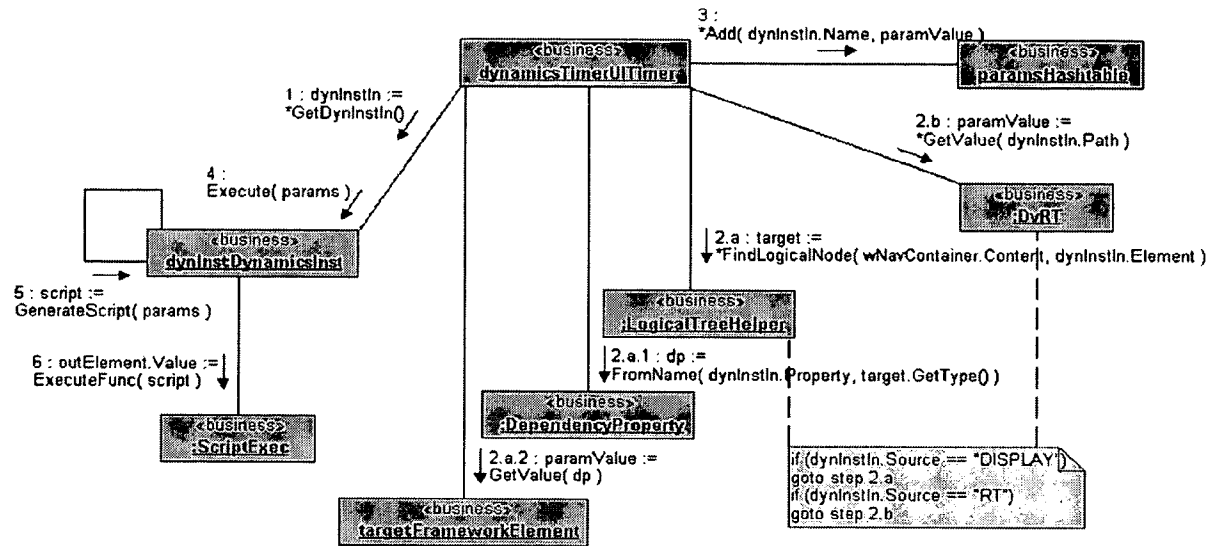


Figure 24 – Object Collaboration Diagram for Dynamics Processing

2.5.5.3 Event Handling

The event handling dynamics process is actually very similar to the processing of Dynamics Instances, except for the fact these dynamics are executed when the associated event is triggered. Once triggered, the generic event handler will go thru the same process discussed in the previous section.

3 Process Modules

3.1 Objectives

The objectives for Process Modules are:

- 1- **Abnormal Situation Prevention** – Users will be able to use Process Module algorithms to determine abnormal conditions that would otherwise be difficult to detect.
- 2- **Integrated** – Process Modules are fully integrated with operator displays, alarms & events, configuration, runtime, historian and other subsystems.
- 3- **Ease of use** – Process Modules will be easy to use. Process Graphic Displays will be able to reference values in Process Modules with very little additional configuration effort. During runtime the Process Module results can be completely hidden or turned on in layers depending on the capabilities of the users and their option settings.
- 4- **Best engineering practices** – The system will be designed to facilitate re-use of processing elements, graphics, and simulations thus minimizing configuration and testing effort.
- 5- **Separation of the process graphics from the process module engine.** The runtime environment for Process Modules will be separate from the engineering and display environment. This is important so that Process Modules can at some point be made redundant, support on-line upgrades, can be easily moved around the system taking advantage of computing resources, etc
- 6- **Flexible data bindings** – Process Modules can retrieve data from multiple data sources including Runtime, Historian, OPC, Services, XML Files, etc.
- 7- **Secure** – Process Modules need to participate in the same session and authentication infrastructure as the rest of DeltaV.
- 8- **Smart Objects** – Process Modules are the algorithms behind Smart Objects technology.
- 9- **DeltaV Look/Feel** – Editors and runtime have the same look/feel as the rest of DeltaV.

3.2 Overview

Operator graphics often depict the devices, equipment and connections between them. The arrangement drawn closely resembles the physical connections in the actual plant. If the devices and equipment on these operator displays were created from Graphical Process Elements representing physical equipment and Process Streams representing physical pipes etc²⁵, then it would be possible to infer additional relationships such as flows, mass balances, etc. The relationships defined in these graphical displays will be stored in the process database²⁶. The following is an example of a Process Graphic derived from the P&ID of a plant.

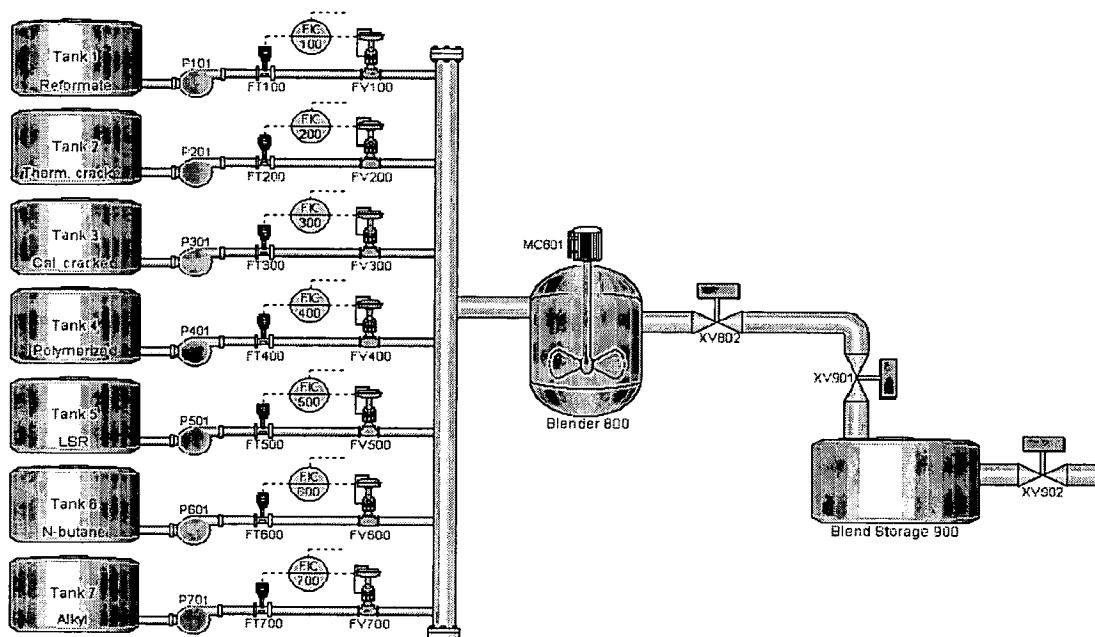


Figure 25. Process Graphic

In the above graphic the Process Element 'Tank' will be aware that it has links to flow transmitters on either end. The awareness is captured in the properties, connectors, algorithm, and rules associated with the Tank graphic.

The underlying process database is updated when the process graphic display is updated and as a result it's business rules are modified. The rules are encapsulated in the process algorithm which has access to the process flow database and to process values from the plant.²⁷

In this example the business rules will cause the tank object to interact with the linked transmitter objects and generate an alarm if it detects that its level is changing when it should not. This may be indicative of a possible leak in the tank, which should be repaired.

²⁵ Traditional operator interface do not automatically embody the logical relationship between these objects; instead each object is shown as an independent entity, and the logical relationships are in the mind of the operator or are embodied thru specially developed logic.

²⁶ Part of Configuration Database

²⁷ The process flow database and rules engine will be co-located. The same process elements and links can be shown on multiple displays.

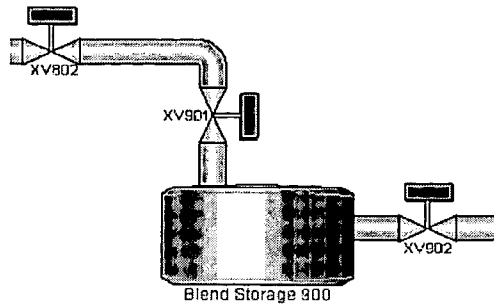


Figure 26. Process Graphic – Advisory Alarms

Using these capabilities it is possible to do the following:

- Abnormal situations are detected and notifications issued.
- Mass balances are calculated for equipment.
- Loss tracking is possible with little or no configuration.
- The interconnection and the logical interaction between equipment provide a higher level of diagnostics for the plant.

In a control system the operator is faced with managing large process areas. Each process area may have hundreds of measurements that indicate the process operating conditions. Thus, it is not possible for the operator to constantly monitor all process measurements. As a support for the operator, process control systems have supported process alarming based on these measurements to alert the operator of an abnormal processing condition. Such alarms may be based on the absolute value of a measurement or on the deviation of a process parameter from setpoint. However, since a process may be required to run over a wide variety of operating conditions, process alarm limits may detect extreme cases of sudden, complete failure.

Finding an effective means of providing early detection of measurement or process failure has been a key area of research in academia and industry over last 20 years. Much progress has been made in the application of multivariate statistical process control, MSPC, for detecting and isolating process and measurement faults. By leveraging off of the results of this research, an easy to use MSPC capability may be embedded in the DeltaV control system. DeltaV customers can use the MSPC capability to improve plant performance by reducing loss production and off-spec. Thus, multivariate analysis based on MSPC capability can differentiate DeltaV from other control systems that do not have this capability.

Each of these capabilities is supported by standard framework called Process Modules. Each specific application requires an algorithm type. The algorithm types that will be discussed in this document are:

- 1- Process Simulation
- 2- Expert
- 3- MSPC
- 4- Other (e.g. Mass balance, Optimization)

3.2.1 Process Simulation

The implementation of process simulation is an important part of the Starburn project. Many of the basic concepts associated with the creation of simulation based on the operator graphic display were disclosed in the Starburn concept document. The information below presents an introductory background. Full details are provided in the Process Simulation Concept Document.

3.2.1.1 Processing Elements

A set of predefined graphic elements will be provided for the construction of Process Graphics and Process Modules. These graphic elements will be designed to dynamically show on-line measurements and actuators that interface to the DeltaV control system. Unmeasured parameters that reflect process operation will be calculated using on-line process simulation – these calculated parameters will be shown as an integral part of the associated graphic elements.

In an offline environment used for engineering or training environment, the graphic elements will automatically show measurement values based on process simulation. The values calculated by process modules are based on the actuator position or state as well as manual disturbance values. Process modules can be created from the graphic elements used to construct Operator Process Graphic Displays.

3.2.1.2 Process Connections

In many processing plants, the step associated with manufacturing may involve handling solid materials, liquid and vapor, and gases. To clearly illustrate the material flow through the process, three different types of process connections will be support.

- Piping – for liquid and high pressure vapor or gas flow
- Duct – for low pressure gas flow
- Conveyor – for the movement of solid material between processing units.

The properties of the material that is transferred by connections are determined by the upstream input. This information plus the *connection status* are available as properties of the connection. The elements that may provide this upstream input are the following:

- Processing element *Output*
- Actuator element *Output*
- Stream element *Output*

The properties of a connection will be displayed when the cursor is placed over the connection. Also, the properties associated with a connection may be exposed for permanent display by placing a measurement or estimated property element on the connection.

A connection element may terminate at one of the following elements:

- Processing element *Input*
- Actuator element *Input*
- Stream element *Input*

3.2.1.3 Process Actuators

Actuators may be placed between process connects or between processing elements and a connection. In some cases an actuator may be used with a specific connection type i.e. pipe, duct or conveyor, as defined below:

	Pipe	Duct	Conveyor
--	------	------	----------

Regulating valve	X		
On-Off Valve	X		
Pump	X		
Eductor	X		
Force Draft Fan		X	
Induced Draft Fan		X	
Damper Drive		X	
Feeder	X		X
Motor Drive			X

3.2.1.4 Measurements and Property Elements

The transmitter element is used in the display to access the measurement value associated with a physical transmitter or switch. This element can be added to a connection element or to a processing element. When a transmitter element is added to the display, the user must identify the associated AI, PCI or DI block in DeltaV. In the on-line mode, the value of the measurement will be shown next to this measurement element. In the off-line mode the simulated value of the measurement will be automatically displayed based on the value calculated by the associated process simulation module.

An estimated property element may be added to a connection or processing element to display any property of that element. When this element is placed on a connection or on a piece of equipment then the user can browse and select the properties that will be displayed. Thus, properties that are not available through a physical measurement may be exposed through the use of the estimated properties element.

3.2.1.5 Processing Elements

Plant equipment other than measurements and actuator and connection will be represented in the operator display as processing elements. All inputs and outputs to a processing elements will be made through connection elements.

Standard processing elements will be provided for the following plant equipment:

- Tank – vertical
- Tanks – horizontal
- Heater
- Static Mixer
- Reactor
- Mixer
- Air Heater

For these standard processing elements, the user may specify the number of inputs and outputs to and physical equipment properties e.g. size. The simulation algorithm and static representation of these standard processing elements may not be modified by the user.

All other plant equipment will be implemented as **Custom Processing Elements**. The static representation, number of inputs and outputs and the simulation algorithm may be modified to meet the user interface requirements. Once a custom processing element has been defined, it may be saved as a template that may be reused or used as a starting point in the creation of other processing elements.

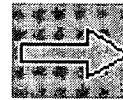
Custom Processing Elements such as distillation columns, evaporators, separators, boiler etc may be described using a step response model relating each input to each output of the vessel. Inputs may be gas and/or liquid streams. Optionally, the user may define the equations that describe the relationships between the inputs and outputs of the processing element.

3.2.1.6 Process Graphic Streams

Stream elements may be included in a display to define the starting properties associated with a connection element. Also, stream elements may be used as connection points between displays. For such off-sheet connections between displays, the user may click on the stream to immediately call up the associated display that contains the referenced connection.

3.2.1.6.1 Mass / Composition

The mass/composition stream element will normally be used for define the starting properties of a process input i.e. the starting feedstock composition, etc. or to define a link to a stream connection on another display. Connections may be made on the input or output of the mass/composition steam element (not both)

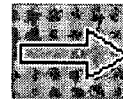


The following must be configured for this element:

- Name of the stream (unique within the DeltaV System)
- Properties of the steam (if no reference input of input connection)
 - Mass fraction of component (if more than one)
 - Pressure or mass flow
 - Temperature
 - Specific heat
 - Density
 - Connection type (pipe, duct, conveyor)
- Referenced Input Stream(if used of accessing a stream on another display)

3.2.1.6.2 Energy

The energy stream element will normally be used for define the starting energy associated with a process input i.e. the BTU/HR transfer, etc. or to define a link to the energy properties of stream connection on another display. Connections may be made on the input or output of the energy stream element (not both).



The following must be configured for this element:

- Name of the stream (unique within the DeltaV System)
- Properties of the steam (if no reference input of input connection)
 - Energy flow
 - Connection type (pipe, duct, conveyor)

- Referenced Input Stream(if used of accessing a stream on another display)

3.2.1.7 Example - Fluid and Gas Flow Simulation

In Fluid and Gas Flow Simulation pressure is the driving force for liquids or gas transfer between vessels. The flow through a transfer pipe or duct is based on the starting pressure and the pressure drops introduced by the processing elements in the flow path e.g. pipe, valves and the point of discharge. To illustrate this three typical examples that will be encountered in a process simulation are presented below.

Specified Starting Pressure – In some cases, the upstream equipment may not be included in the simulation and the user must specify the starting pressure as part of the stream properties – pressure is defined as the independent parameter. When stream pressure (starting pressure) is specified, then the flow through the associated pipe is determined by the pressure drop introduced by the pipe and the elements in the pipe and the pressure at the point of discharge into a tank, as illustrated below.

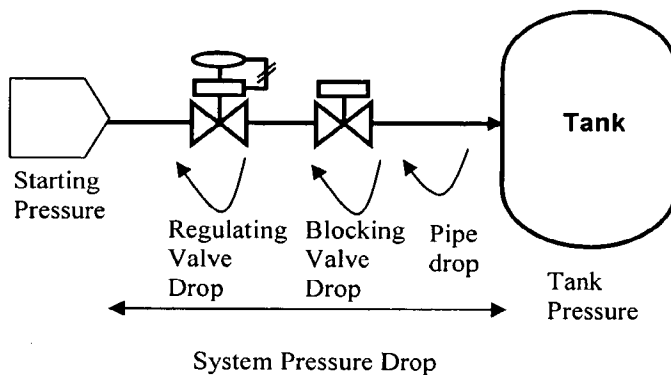


Figure 27. Fluid and Gas Flow Simulation – Specified Starting Pressure

Vessel Establishes Starting Pressure – If a vessel is pressurized, then this pressure may be used as the driving force for transfer of vapor to liquids between tanks or from a tank to a vent, header, etc.

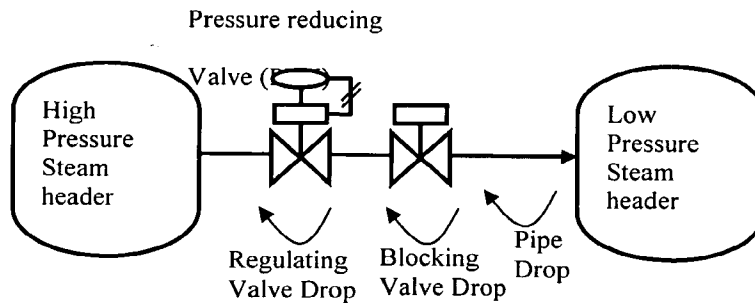


Figure 28. Fluid and Gas Flow Simulation – Vessel Establishes Starting Pressure

For this example, the driving force for steam flow is the differential pressure between the high and low steam header. Since the pressure in each header, then the differential pressure is variable. The pipe flow thus is determined by this differential pressure, the position of the regulating and on-off valve, and line loss. This relationship is similar to that previously illustrated for the case where the starting pressure was specified.

Pump or Fan Establishes Starting Pressure – A pump or fan is often the pressure source that provides the driving force for liquid or gas flow respectively. In such cases, the pressure provided will be a function of the flow. This relationship may be documented by the equipment manufacturer by a pump or fan curve that shows the expected pump differential pressure as a function of pump flow. The following variable speed pump example may be used to illustrate the impact of pump discharge pressure with flow rate. In this example, the liquid supply to the pump is assumed to be from a tank at constant (atmospheric) pressure. In this example, pump discharge flow travels through a pipe to a heat exchanger before entering another tank (at atmospheric pressure) as illustrated below.

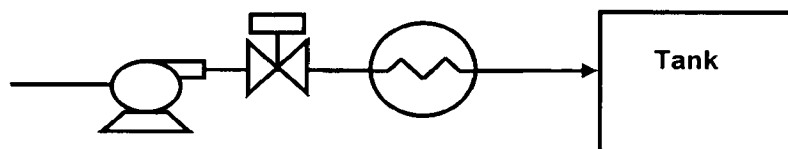


Figure 29. Fluid and Gas Flow Simulation – Pump or Fan Establishes Starting Pressure

3.2.2 Expert

3.2.2.1 Background

Conventional programming languages, such as C#, are designed and optimized for the procedural manipulation of data (such as numbers and arrays). Humans, however, often solve complex problems using very abstract, symbolic approaches which are not well suited for implementation in conventional languages. Although abstract information can be modeled in conventional programming languages, considerable programming effort is required to transform the information to a usable format.

One of the results of research in the area of artificial intelligence has been the development of techniques which allow the modeling of information at higher levels of abstraction. These techniques are embodied in languages or tools which allow programs to be built that closely resemble human logic in their implementation and are therefore easier to develop and maintain. These programs, which emulate human expertise in well defined problem domains, are called expert systems. The availability of expert system tools, such as CLIPS, has greatly reduced the effort and cost involved in developing an expert system.

What is CLIPS?

C Language Integrated Production System (CLIPS) is an expert system tool which provides a complete environment for the construction of rule and/or object based expert systems. Created in 1985 at the Johnson Space Center, CLIPS is now widely used throughout the government, industry, and academia.

Rule-based programming is one of the most commonly used techniques for developing expert systems. In this programming paradigm, rules are used to represent heuristics, or "rules of thumb," which specify a set of actions to be performed for a given situation. A rule is composed of an *if* portion and a *then* portion. The *if* portion of a rule is a series of patterns which specify the facts (or data) which cause the rule to be applicable. The process of matching facts to patterns is called pattern matching. The expert system tool provides a mechanism, called the inference engine, which automatically matches facts against patterns and determines which rules are applicable. The *if* portion of a rule can actually be thought of as the *whenever* portion of a rule since pattern matching always occurs whenever changes are made to facts. The *then* portion of a rule is the set of actions to be executed when the rule is applicable. The actions of applicable rules are executed when the inference engine is instructed to begin execution. The inference engine selects a rule and then the actions of the selected rule are executed (which may affect the list of applicable rules by adding or removing facts). The inference engine then selects another rule and executes its actions. This process continues until no applicable rules remain.

3.2.2.2 DeltaV Expert

DeltaV Expert is an expert system application for process control. It utilizes CLIPS as its core inference engine. This is illustrated in the diagram below:

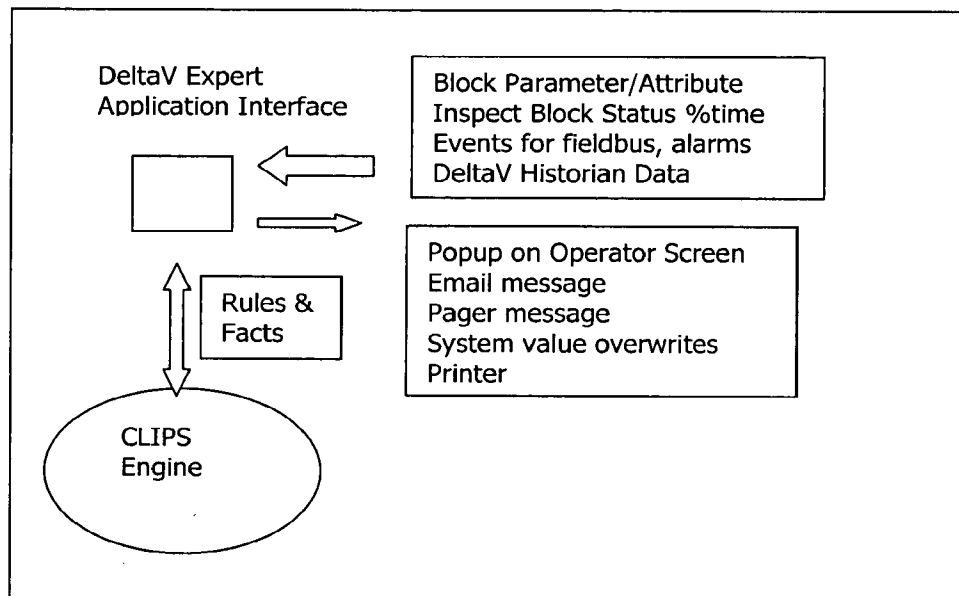


Figure 30 – DeltaV Expert Overview

The idea is to use expert system technology to assist the operator in the detection and management of abnormal situations in a plant. Through the early detection of abnormal conditions using expert systems, it is possible to improve plant operation or to prevent a failure that could cause a process shutdown. Also, during shutdown and startup conditions, expert systems can be used to draw the operator to abnormal situations that might affect plant operation.

Some of the areas of abnormal situation management that an expert system can effectively address are the following:

- Alarm screening – presentation of the root source of an alarm condition so that the operator is not distracted from the critical events that are associated with and equipment failure.

- Detection of process change – Degradation of equipment performance because of improper equipment setup and wear or buildup in pipes and heat-transfer surfaces may alter the behavior of equipment. The automatic detection of significant process changes helps prevent equipment damage and associated production loss.
- Detection of abnormal conditions – The automatic detection of patterns in plant operation that indicate an abnormal condition may bring this information to an operator's attention sooner than process alarms.

3.2.2.3 Integration with DeltaV

Integration of Expert with configuration and runtime infrastructure allows users to configure, document, and then to download an expert system subsystem running on a ProfessionalPlus or Application Station node. Standard fact templates will be provided to allow consistent and easy access to real time data, historical data, event chronicle data, and historical data. To make it easy for a customer to begin using and getting benefit from the expert system capability, predefined expert rules will be provided. The option is provided for an expert to define their own facts and rules to address the unique situation of a particular plant. In addition, DeltaV Expert allows you to examine the execution of the rules application. Fact data that is based on process measurements may be examined as the application executes and, if simulation is enabled, values provided for testing the expert rule evaluation. Also, using the tools provided by DeltaV Expert, it will be possible to place break points in rule evaluation and to examine the state of rule evaluation. This document will be used in developing the use cases for the off-line (configuration) and the on-line (run time) interaction with the rule evaluation and the operator interface support.

3.2.3 MSPC and PCA

Within the discrete parts manufacturing segment of industry, univariate statistical methods, SPC, have been successfully applied to detect deviation of the process from normal conditions. However, in the process industry, such techniques do not generally apply since most process measurements usually demonstrate a strong correlation under normal conditions. However, many researchers have realized that these measurement correlations provide a necessary redundancy to detect, identify and reconstruct a faulty sensor. Two statistical projection techniques have been developed to address the requirements in the process industry; Principal Component Analysis (PCA) (Jackson, 1991) and Partial Least Squares (PLS) (Geladi and Kowalski, 1986). Most of the commercial products for multivariate on the market today have their roots in these techniques.

Two statistics are commonly used in conjunction with MPSC in fault analysis:

- Square Prediction Error Q – a measure of lack of model fit
- Hotelling's T^2 – a measure of the variation within the PCA model

MSPC could be structured as an integral part of DeltaV. By embedding MSPC as a feature of the Process Modules, then the configuration effort could be greatly reduced.

- The engineer could define and generate MSPC from the **operator screen**
- Correlated parameters were automatically known from the process connections on the operator graphic.

The engineer could elect for all correlated parameters in the module to be automatically included in the PCA or the user could elect to include a portion of the equipment in the analysis. In response his request, a historic plot of these parameters would be displayed and the engineer could select the time frame to be considered in generation of the PCA. The PCA would then be automatically generated.

Once the PCA is created, then the pattern of square prediction error can be used to detect a faulty sensor. The faulty sensor can be automatically highlighted in the graphic measurement element on the operator screen – no extra configuration required since the PCA is part of the module and the measurements are defined by the graphic configuration. An instance of an expert kernel could be included with the PCA as part of the process module. The expert kernel could be used to analyze the alarms associated with the measurements associated with the process module.

From the process connections, the cause and effect could be automatically determined in process alarming.

Automatically screen operator alarms other than that associated with the root problem. Module options would be provided to allow smart alarm screening to be enabled or disabled. Default rules used by the expert kernel would automatically be provided to analyze the square prediction error (SPE) associated with the module PCA to detect abnormal processing conditions. The Facts used in these rules would define the range of the SPE for a specific condition. The Facts range value could be established by the engineer viewing the historian data for the parameters in the module PCA. When engineer identifies the time of that an abnormal condition existed, then the SPE value will be automatically saved as the Fact range values for the identified condition. The Fact range values could be established by an operator when he identifies an abnormal condition has occurred during normal operation.

When the expert kernel detects a pattern that matches the facts, then the operator would be notified of the abnormal condition being detected.

The interaction of the PCA, expert kernel with the process module is illustrated below.

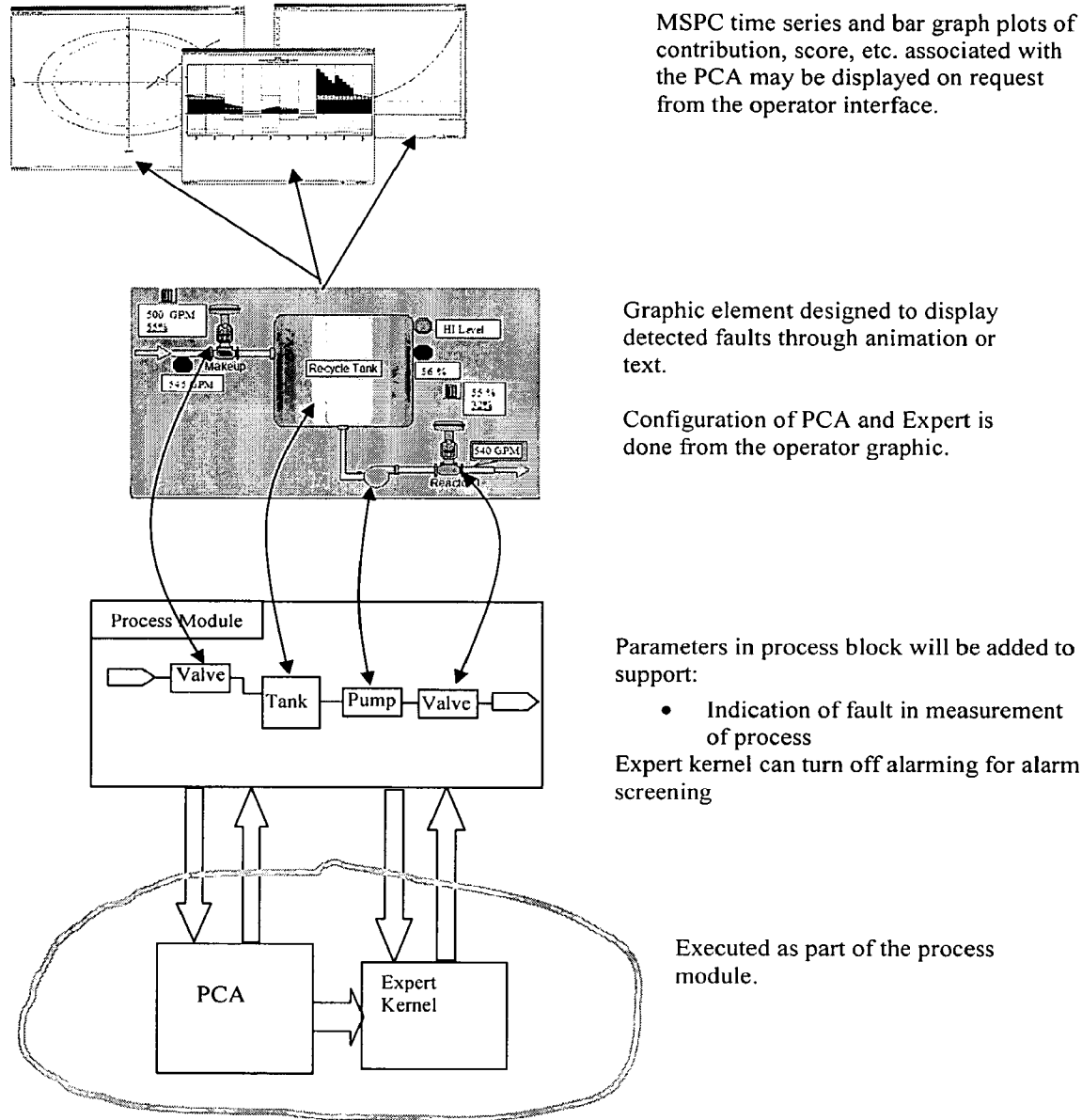


Figure 31. Interaction of the PCA, Expert kernel with Process Modules

3.2.4 Other Algorithms

Other algorithms under consideration are for Route Management, Mass Balance, and Optimization.

3.3 Key Requirements

3.3.1 Process and Processing Elements

- 1- Must be tagged (tagging means that they can be referenced by the runtime).
- 2- Are associated with physical items in the plant – e.g. tanks, pumps, etc.
- 3- Consist of parameters, connection points and behavior.
- 4- Parameters can be simple parameters or smart parameters. Smart parameters can be interpreted and used by the process flow rules engine.
- 5- Connection points represent points at which an object can be attached to other objects by means of pipes or other link mechanisms. A connector is tied to an object parameter.
- 6- Contain Mode, Status, and Alarms.
- 7- Behavior can be coded in a language such as C#.
- 8- Can also have rules associated with them.
- 9- Are arranged in Class Libraries (e.g. a pump).
- 10- Can be collected together into a composite structure. The composite structure would be a Class Based item in the Class-based hierarchy.
- 11- Behavior is executed by “Display Module Execution Engine”.
- 12- Rules are executed by a separate rules engine.

3.3.2 Streams

- 13- Must be tagged.
- 14- They are a representation of physical pipes and other link mechanisms.
- 15- Are used to represent process flow between “smart” process objects
- 16- Contain special properties that define how different “things” flow through the smart links (e.g. steam, electricity, water, sewage, etc.).
- 17- Ensure that Engineering Units of the source and destination object connectors match. In the runtime they also provide opaque data conversions.
- 18- Can be debugged.

3.3.3 Process Algorithms

- 19- Have special properties including, Mode, Status, and Alarm Behavior.
- 20- Can be assigned to workstations.
- 21- Are downloaded as part of display downloads.
- 22- Can simultaneously execute one or more process flow algorithms types: Mass Balance, Routing, Efficiency, Optimization, Economic Calculations, other.

- 23- Can read parameters from DeltaV Control Modules.
- 24- Can expose parameters for access by other DeltaV Control Modules.
- 25- Can control the execution of process flow algorithms. Execution can be enabled and disabled.
- 26- Are verified before download in the process flow database.
- 27- Can be acquired and released.

3.3.4 Process Modules

- 28- Have runtime execution.
- 29- Alarming

3.3.5 Documentation

- 30- Process Elements represented in the Process Graphics may also provide hot links to key documentation.
- 31- The documentation may be applicable to the type of object, or may be specific to the instance (depending on the criticality) of the device.
- 32- The documentation may be vendor supplied.
- 33- The documentation may be user supplied.
- 34- The user will be able to add/delete/change documentation independent of the system software.
- 35- Documentation may include configuration, operational and maintenance categories.
- 36- Clicking on the object brings up the instance specific (if any) and generic documentation.

3.3.6 Expert

- 37- .

3.3.7 MSPC and PCA

- 38- .

3.3.8 Other

- 39- .

3.3.9 Documentation

- 40- Process Elements represented in the Process Graphics may also provide hot links to key documentation.

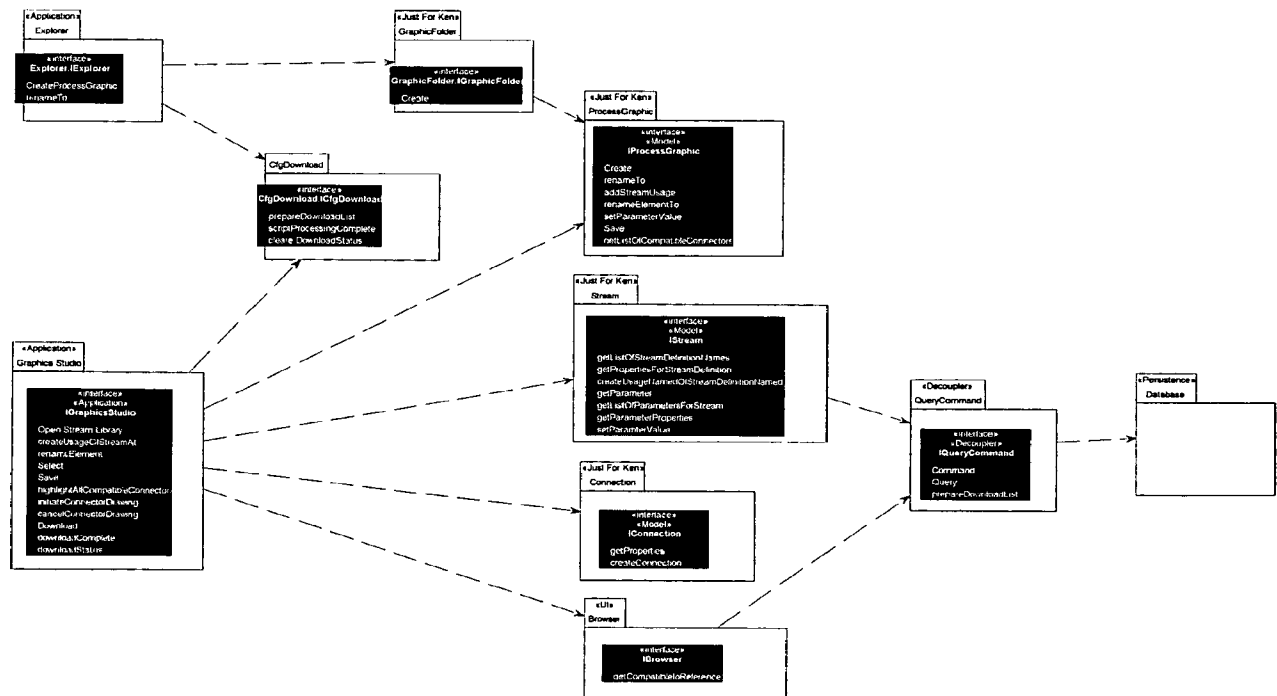
3.4 Process Module Architecture

3.4.1 Overview

3.4.2 Configuration

3.4.2.1 Overview

The...



3.4.2.2 Simulation

3.4.2.2.1 Generation of Process Modules from Process Graphic Displays

The configuration environment for Process Graphics and Process Modules will be integrated with the existing configuration environment to ensure that configuration is entered once²⁸. Display configuration includes operator displays, dedicated displays for batch and advanced control, dedicated displays for alarm and alert management, dedicated displays for route management, etc. The Process Graphic Display configuration environment is illustrated in the figure below.

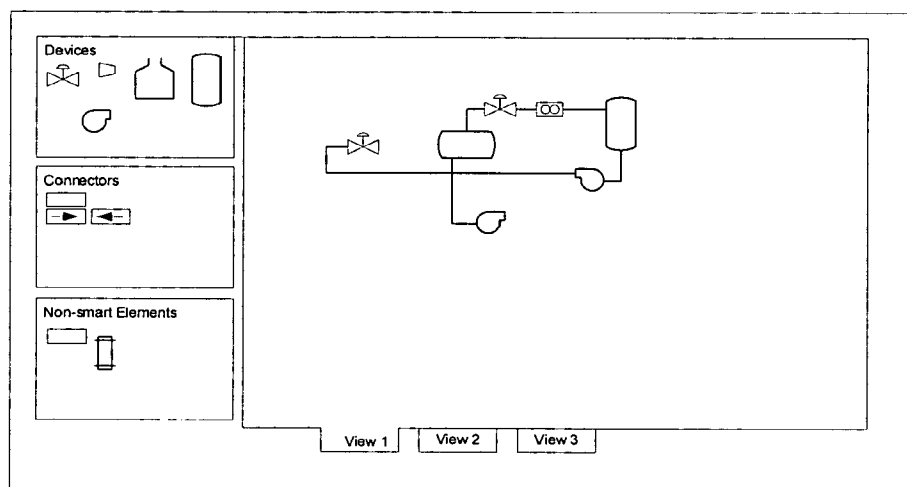


Figure 32. Configuration Environment

At the Process Graphic level we have a view that is very similar to what is available with iFIX today. At this level the user selects Process Graphic elements such as Valve, Pump, Tank, Heater, Mixer, etc drops them on a Process Graphic and then connects them together using piping. The user adds animations, dynamic values, etc to control, measurements or actuators as they would with iFIX.

At configuration time users can select Process Graphic Displays and generate process simulation. The dynamic simulation will be constructed from simulation function blocks that are based on simple composition calculations, mass balance, energy balance and custom calculations.

If high fidelity process simulation is required, then a simulation package such as HYSYS may be linked into DeltaV through process modules. In this manner, information from the process simulation may be made available on the Process Graphic Display.

A Process Module may be generated from a process graphics. The functionality available to the Process Module is determined by the Process Graphic elements. What should be clear is that the Process Module will be constructed to shadow the Process Graphic Display.

²⁸ The basis for the integrated environment was covered in the Common Control Language Patent which issued in 1997. What wasn't thought of at that time was using the integrated environment as discussed here to embed functionality.

When the user configures their Process Graphic Display they have the ability to include additional information such as Mass or Energy Streams. These streams are used in the process module to establish starting conditions needed by the simulation function blocks.

Since Process Modules are real DeltaV Modules, it is also possible for them to reference, and be referenced by, DeltaV parameters, control strategies, displays, etc as illustrated below. Also, using this capability, it is possible for a process module to be created in DeltaV Control Studio independent of the Process Graphic Display.

3.4.2.2.2 Process Modules and Process Displays for Off-line Simulation

The process simulation may be designed to include hand valves and local panel inputs that will exist in the plant but are not wired into DeltaV. Traditional operator training systems (OTS) have included custom built instructor graphic displays that can be included graphic displays to represent and provide access to these local components. To avoid the need for separate instructor displays, the operator graphic will be designed to support layers of information that may be visible or not visible depending on the user and whether the display is being used for off-line simulation or on-line operation. In particular, it will be possible for the user to add an instructor layer to any operator graphic. Through this instructor layer, it will be possible to add manual valves and other graphic elements that represent and provide access to the simulation of local valves, local panel input, and disturbance inputs to the process simulation.

In the graphic display editor, a number of display pallets to support the creation of graphic displays. The following standard palletes provided in DeltaV:

- Calculation and Control – elements to access information from function blocks used in control and calculation e.g. PID function block SP, PV, OUT
- Properties and Measurement– elements used to access or specify field measurements and simulated properties associated with connections or process equipment i.e. field measurement, simulated values.
- Actuators – elements representing field devices used to set or regulate process streams
- Processing – elements representing common process equipment.
- Custom – elements that allow equipment that is specific to a manufacturing process to be represented in the graphic display.

As in Control Studio, the user may create his own palette from items included in these standard palletes. All the elements included in the display except for Steam and Connection elements may be modified by the end user to satisfy his display requirement.

3.4.2.3 Expert

3.4.2.4 MSPC and PCA

3.4.2.5 Other

3.4.3 Runtime Execution

3.4.3.1 General

3.4.3.2 Simulation

The control environment for Control and Process Modules will allow access to parameters, alarms, etc. This integrated environment control environment is illustrated in the figure below.

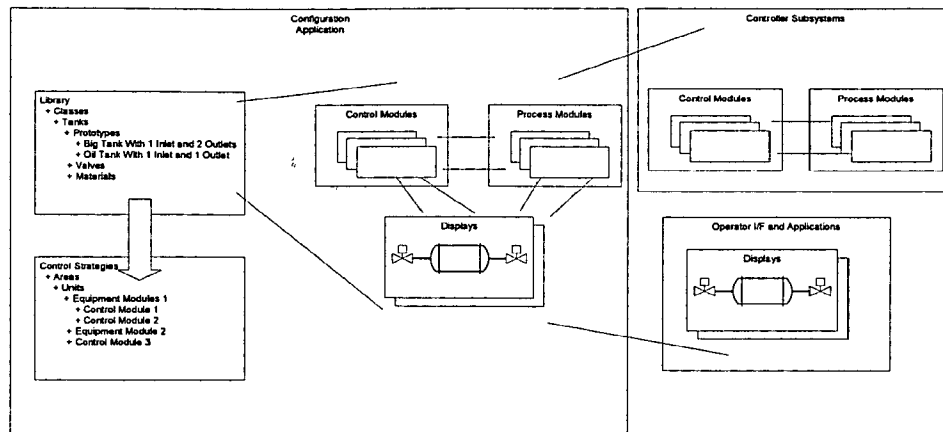
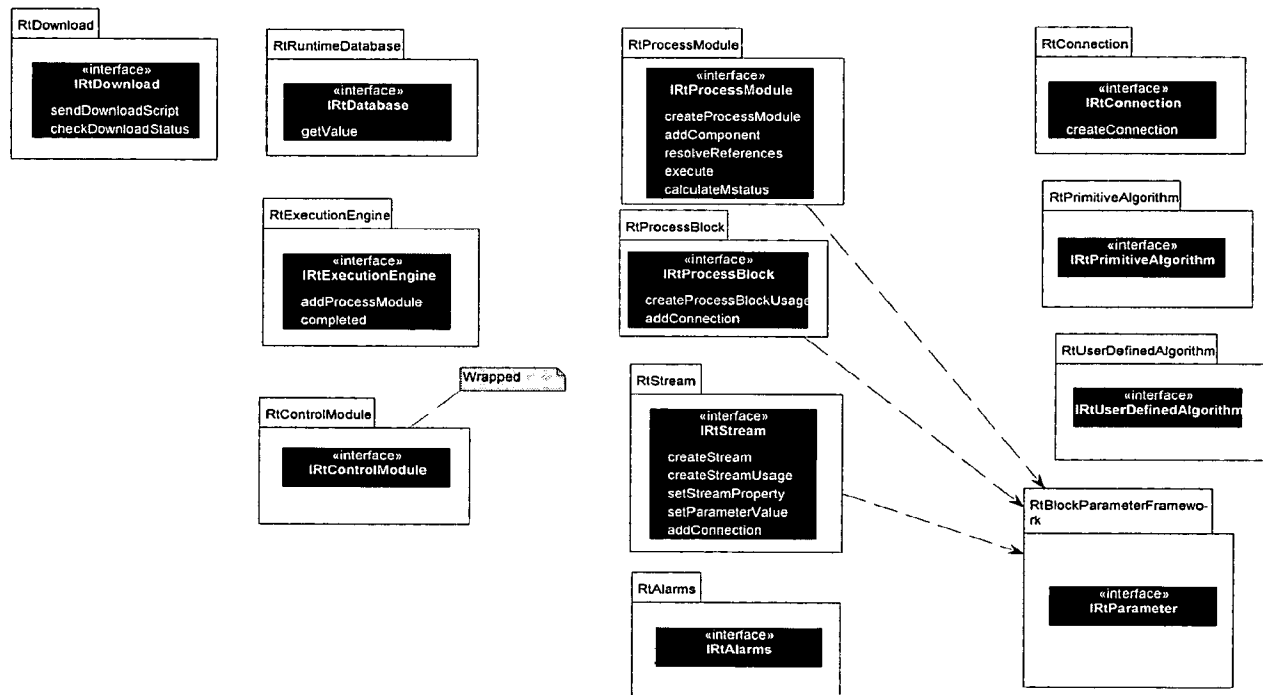
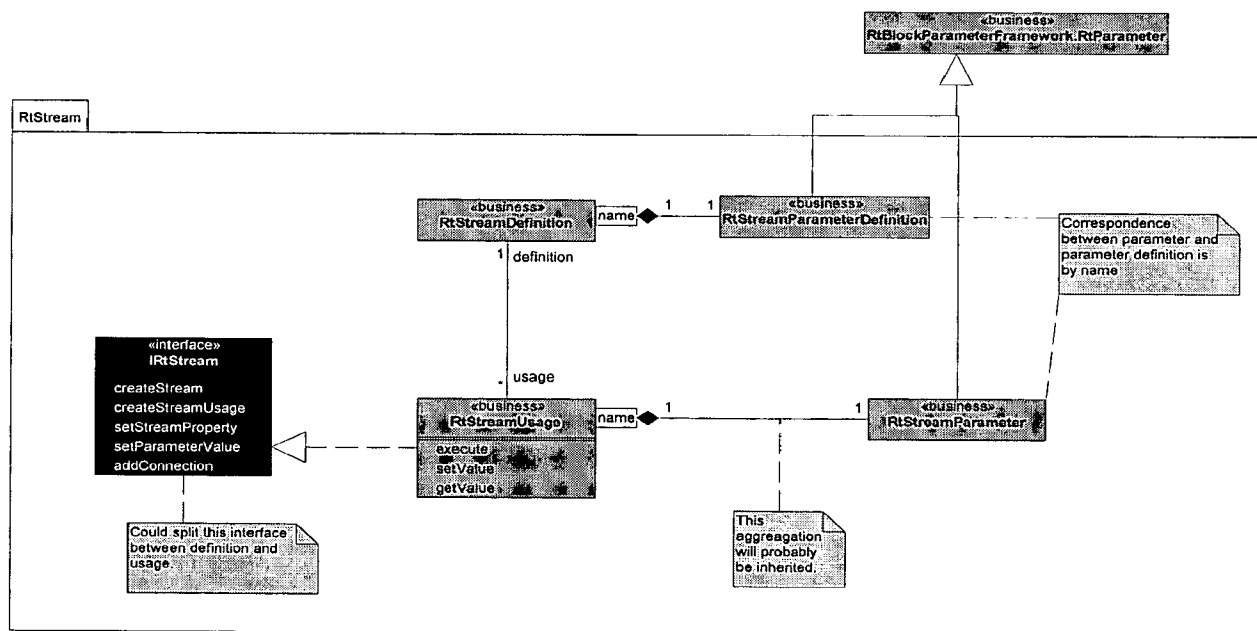
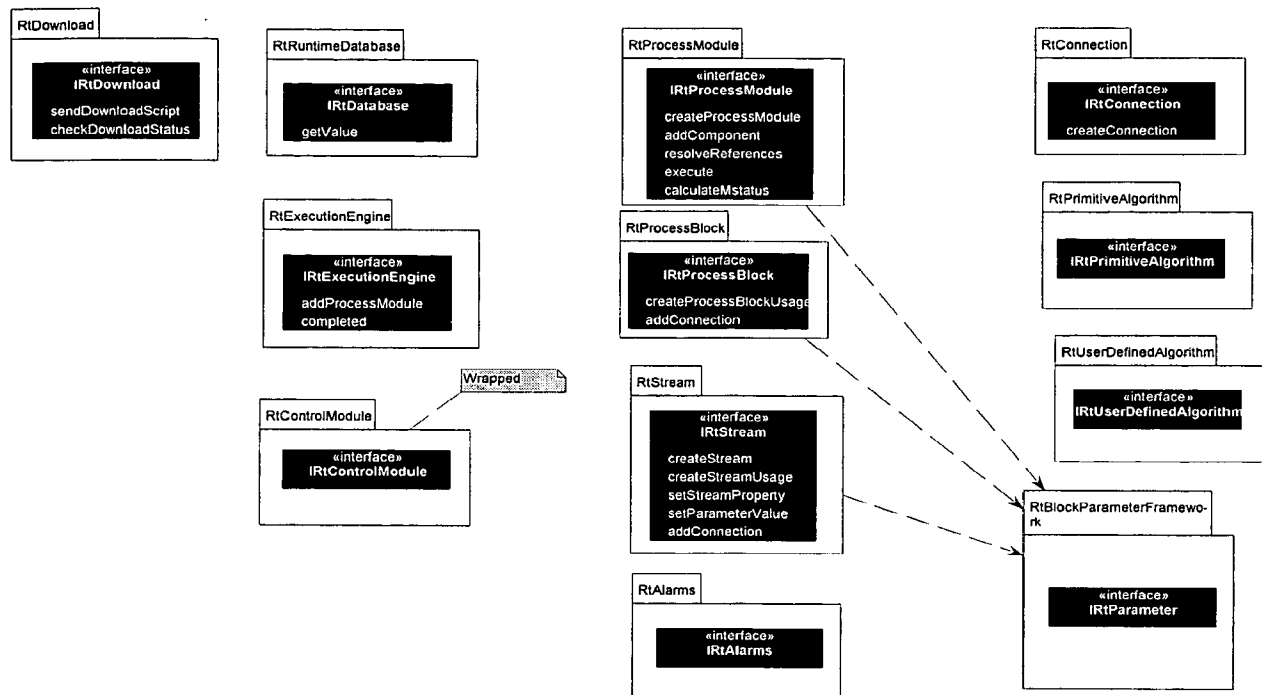


Figure 33. Integrated Control Environment





3.4.3.3 Expert

3.4.3.4 MSPC and PCA

3.4.3.5 Other

3.4.4 Operator Displays and Alarming

3.4.4.1 General

3.4.4.2 Simulation

3.4.4.3 Expert

3.4.4.4 MSPC and PCA

3.4.4.5 Other

3.4.5 Diagnostics

3.4.5.1 General

3.4.5.2 Simulation

3.4.5.3 Expert

3.4.5.4 MSPC and PCA

3.4.5.5 Other

4 Runtime Workspace

4.1 Overview

The runtime workspace provides a completely integrated environment for operators. Although the primary applications are Process Graphics and Alarm Management, the workspace must also host other applications such as trending, batch, and advanced control. It is becoming common to also host other applications such as streaming video.

4.2 Objectives

The runtime workspace and integrated applications are the main operator interface for the DeltaV System. The primary objectives for the runtime workspace are:

1. Highly reliable application intended for process control operators who need to stay in control of the manufacturing process in all circumstances. Characteristics include:
 - Always available. In many cases, the application must be available for long periods of time (e.g. weeks, months, even years) without unplanned disruptions.
 - Always interacting. Operator must always have the opportunity to "move on"; getting predictable responses to new directives even when previous requests take "too long" to complete, or "everything is changing" due to a plant upset.
 - Mistake resistant. Repeatable behaviors and user interaction features that can help reduce the possibility of high cost "human errors"; especially in high stress situations.
2. Extremely responsive interactions and high performance display of process information. Expectations for large user configured displays to appear and be populated with live data from the DeltaV system typically within 1 second. (Smaller displays like our pre-engineered "faceplate" style displays correspondingly faster.)
3. Supports a dedicated (kiosk style) operator environment, which is extremely resistant to users damaging the programs or data present on that workstation, or gaining access to unintended applications. The degree of restriction is user configurable, so that the workspace is also suitable for use by trusted and higher skill level users.
4. Adaptable to a wide range of user interface hardware features:
 - 1- 2- 3- or 4- monitor video configurations
 - 4:3(PC format) and 16:9(HDTV format) aspect ratio monitors
 - mouse + keyboard operation
 - touch screen operation
 - PDA form factors and user input techniques
 - Tablet PC form factors and user input techniques
 - Smart Phone form factors and user input techniques
5. Excellent integration of operator tools/applications (e.g. Batch, Advanced Control, Diagnostics, History viewing) with user configurable displays (and user display logic). Easy access to information related to display elements. Easy access to auxiliary information sources (e.g. video feeds) and collaboration technologies (e.g. e-mail and instant messaging.)
6. Operator-friendly tools to select and manage the content in the runtime workspace; minimizing window management responsibilities. Improved features for rendering content within the available space, so that a content sources not specifically engineered for the workspace are easily usable.

7. Well integrated user security and authentication facilities to support:
 - Fast DeltaV user switching (comparable to pre-Starburn systems)
 - DeltaV user collaboration (temporarily aggregating security privileges)
 - Multi-user authentication for electronic signature requirements
 - Authentication technologies that do not require passwords
 - Login timeouts
8. Platform for improved information capture and analysis tools for operations staff. Features like:
 - Snapshot log: recording information for later examination
 - Tools for finding revealing alarms in an alarm flood
 - "Back in time" displays: ability to look at displays based on (automatically collected, recent) historical data

4.3 Workspace Desktop Management

The runtime workspace application(s) will operate in either of the following desktop management modes:

- The "Dedicated and Controlled" desktop
- As just "Another Windows Application"

4.3.1 Dedicated and Controlled Desktop

The "dedicated and controlled" desktop management mode allows DeltaV system administrators to deploy the runtime workspace so that its users do not inadvertently (or maliciously) damage the software/data installed on that (serving) workstation, nor cause the runtime workspace to be inoperative.

4.3.1.1 User needs

- a) It is possible to configure a workstation (or session on a server) to automatically start after system boot-up in "dedicated and controlled" desktop management mode. In this mode, one instance of the runtime workspace is allowed to run in each workstation (session).
- b) A "workspace hard reset" mechanism is required, which causes the run-time workspace to revert to the initial startup condition (framework contents) without requiring it to be shut down and manually restarted. This is intended as the last resort available to operators to restore an (apparently) malfunctioning runtime workspace to working condition (short of rebooting the workstation/server). It is acceptable for a "workspace hard reset" to take up to 30 seconds²⁹.

During the reset, a message should be provided indicating that a reset is in progress, along with some updating indication that the screen hasn't frozen.

- c) Certain users should not be constrained to the "dedicated and controlled" desktop management mode. Users with appropriate DeltaV security keys, can switch to "another Windows application" desktop management mode (retaining as much current workspace context (panel contents, recently used history, etc.) as possible).
- d) While in "dedicated and controlled" desktop management mode, users are prevented from running unintended programs or applications. This requires:
 - Disabling access to the Start dialog (Windows key and Ctrl-Esc)
 - Disabling access to the Windows taskbar
 - Disabling access to Windows keyboard shortcuts; especially including:
 - a. Run dialog (WinKey + R)

²⁹ The intent that this would be a "deep" enough reset to completely clean up and reinitialize the runtime workspace and all runtime workspace companion applications.

- b. Minimize all (WinKey + M)
 - c. Switch to another (non-runtime workspace) application (Alt-tab)
 - d. OS Explorer (WinKey + E)
 - e.
- Not allowing access to Windows desktop shortcuts
- e) The runtime workspace will allow specific (non-runtime workspace) applications that have been authorized (via DeltaV configuration). The user will be presented a list of applications (and descriptions) to choose from³⁰.
- f) While in "dedicated and controlled" desktop management mode, users are prevented from making the runtime workspace inoperative. This requires:
 - Not allowing the termination of the (primary) runtime workspace application (e.g. Alt-F4, or Exit menu items)
 - Disabling access to the Windows Security dialog (Ctrl-Alt-Esc)
 - Disabling access to Windows keyboard shortcuts; especially including:
 - a. Minimize all (WinKey + M)
 - b. Lock workstation (WinKey + L)
 - c.
 - Disabling access the Windows Display Properties dialog (e.g. access to altering display color depth and resolution settings, appearance preferences, themes, wallpaper, etc.)
- g) The runtime workspace will provide a "constrained Internet browser". While in "dedicated and controlled" desktop management mode, web pages from authorized (via DeltaV configuration) URLs will be displayed or accessible via hyperlinks³¹.
- h) While in "dedicated and controlled" desktop management mode, any screen savers are disabled.
- i) Runtime workspace compliant applications do not provide access to the Windows folder and file properties dialog when performing any file browsing operations. They do not allow files to have their properties or security requirements changed, or be renamed or deleted, unless doing so cannot possibly damage the software or data installed on that workstation.

4.3.2 Another Windows Application

The "another Windows application" desktop management mode allows the runtime workspace to be used in conjunction with other Windows applications, by appropriately authorized and skilled personnel. Switching to this mode is useful for:

- using the DeltaV system configuration applications (including workspace and display configuration applications)
- gaining access to debugging features not appropriate for all users
- troubleshooting the runtime workspace

4.3.2.1 User needs

- a) To aid runtime workspace troubleshooting, the "another Windows application" desktop management mode supports all the features and interactive behaviors of the "dedicated and controlled" desktop management mode as much as possible. Conceptually, it just removes constraints and adds extra features.

³⁰ Configurers are responsible for choosing non-runtime workspace applications to run that don't allow the operator to alter or delete files, launch still other "unconstrained" applications, etc.

³¹ Goal here is to support users who maintain documents and web pages on intra-network servers; and still want to prevent uncontrolled roaming on the Internet.

- b) While in the "another Windows application" desktop management mode, the user has access to the normal Windows desktop application management features; including:
 - Task bar, Start button, Run dialog, etc.
 - Windows key and Windows key shortcuts
 - Application minimization (including the runtime workspace)
 - Application switching (Alt-tab)
 - Unrestricted ability to change display properties
 - The ability to terminate the runtime workspace application(s)
- c) While in the "another Windows application" desktop management mode, there is a mechanism to "spin off" a new (detached) Windows application window on the Windows desktop, with the same content (display or runtime workspace application) as any of the framework panels (fixed or floating). This gives the user the ability to easily create content in new windows, where s(he) is responsible for window management (positioning, resizing, minimizing/maximizing, scrolling content within the window client area, closing, etc.) Once such windows are created, they no longer interact with or "track" actions in the runtime workspace framework.
- d) With access to the Windows desktop, users may "run" new instances of the runtime workspace application; choosing for it to start up in either "dedicated and controlled" or "another Windows application" desktop management mode:
 - When launched in "dedicated and controlled" desktop management mode, the Windows desktop will be controlled by the runtime workspace application, and other running applications will (probably) be inaccessible³².
 - When launched in "another Windows application" desktop management mode, the user may run multiple instances of the runtime workspace application³³.
- e) All users may switch from the "another Windows application" desktop management mode to the "dedicated and controlled" desktop management mode (no security key is required.) The switch retains as much current workspace context (panel contents, recently used history, etc.) as possible.

4.4 Internationalization Support

Today, most "operations" features of the DeltaV system are available, localized to one of several languages. The runtime workspace, being the primary "operations" application, must continue to support the current language choices. It should also be expected that not all workstation/terminals/PDAs connected to a DeltaV system will want to have the runtime workspace use the same language choice³⁴.

Also, should technical support from EPM/AUS be needed to consult on a non-English DeltaV system, the English speaking technical support staff has the option of switching to, or running a separate English runtime workspace to facilitate communications³⁵.

4.4.1.1 User needs

- a) Users may choose one or more of the available DeltaV language options when installing DeltaV on runtime workspace-capable workstations or terminal servers. One language is the identified

³² The user should probably close (or park) them appropriately before starting up a "dedicated and controlled" runtime workspace.

³³ For example, taking advantage of multi-monitor hardware to use/test several single monitor display frameworks running in different instances of the runtime workspace application. Or perhaps run an English instance of the runtime workspace in parallel with a non-English instance, to assist English speaking troubleshooters.

³⁴ Since most engineering and administrative applications are available only in English, it seems quite likely that some users will want to run English runtime workspaces in non-English DeltaV systems. (Seems like this could facilitate technical support as well.)

³⁵ If this works pretty well, we may want to consider always installing English secondary language support on workstations in non-English DeltaV systems.

as the "dominant" language for the system; determining the standard and default configuration values for that DeltaV system. The dominant language must be the same for all DeltaV workstations in the same system.

- b) When an instance of the runtime workspace is started, the dominant language is automatically chosen by default, and all standard workspace behaviors and interactions use that language.
- c) It is possible to override the language choice for the runtime workspace with additional information provided at start up. It is also possible for a user to switch to another language for an existing runtime workspace instance by selecting one (of those installed on this workstation) in the course of doing a "workspace hard reset" operation.
- d) When the workspace is running using a language other than the dominant language for the system, the workspace applications provide an interface (menus, dialogs, etc.) localized for the chosen language. Information content that is determined by the DeltaV system configuration, such as:
 - Area, module, node, DST, FF device names
 - Module, node, FF device descriptions
 - User enum state names
 - Alarm parameter names, alarm words, priority words
 - Batch procedure/recipe names, grade names...still retain their configured names (presumably compatible with dominant language.)

Also, information stored in common system databases (available to multiple workstations/sessions) such as:

- Event Chronicle databases
- Batch Historian databases
- Continuous History databases
- Device Audit Trails

...remain, and are displayed using the dominant language for the system.

4.5 Workspace Framework

The workspace framework provides a user configurable layout for how information is arranged within the runtime workspace; utilizing the available display area on the target display hardware. We anticipate:

- Users will often choose one of the pre-engineered frameworks that are part of the default DeltaV configuration. Users (or system integrators) with special requirements or unique display hardware will design/configure their own framework(s).
- Once a framework is chosen, it will be widely used within a DeltaV system (across many workstations/terminals having similar display hardware). This helps makes operators be more effective wherever they are working; and helps reduce training time and mistakes.
- Displays will be designed with the framework (panel) dimensions in mind; building displays that will "nicely fit" the panels without a lot of "stretching" distortion or having display elements end up being too big (wasting precious space) or too small (to read easily).
- Even when different display hardware configurations are used in the same system, users will strive to use similarly sized panels across their frameworks, so that the same displays will render nicely on each hardware configuration; avoiding the effort of engineering separate displays for each.

4.5.1 Framework Panels

Framework panels are rectangular display areas where runtime workspace displays or applications can be made to appear. There are two types of framework panels:

- **Fixed panels:** have a fixed location in the framework and area always visible (even if they have no content). Are not allowed to overlap other fixed panels. The collection of fixed panels in the

workspace framework is the visible "background surface" of an instance of the runtime workspace application.

- **Floating panels:** provide a means for a controlled number of temporary content windows that float over (obsuring) other panels. When the content in a floating panel is closed, the floating panel disappears. Floating panels may be configured to be movable: allowing the operator to move the floating panel away from it's anchor point, giving him more control³⁶ over the visible portions of all panels.

4.5.1.1 User needs

- a) Framework panels are capable of showing several types of content at any point in time:
 - User configurable runtime workspace (graphic) displays.
 - Runtime workspace applications.
- b) Floating panels in the workspace framework have a configurable "use order". When two closed floating panels are otherwise equally eligible to be selected to display new content, the floating panel with higher precedence in the use order will be chosen.

Fixed panels have a separate configurable "use order" to establish precedence.

- c) Through configuration, it is possible to categorize workspace display definitions and framework panels so that displays of a certain category will be automatically directed to appear in a fixed or floating panel with a matching category name³⁷ when a destination panel is not otherwise specified.
 - One framework panel (fixed or floating) may be configured with 0 or more category names.
 - A display definition may be configured with 0 or 1 category names.
- d) It is possible to assign the same category name to several (fixed or floating) panels. If a new display's category matches several panels, the display is directed per the sequence:
 - If a fixed panel that currently has no content matches the new display's category, the the new display will replace the old content in that fixed panel. Fixed panel use order will resolve which fixed panel to use if more than one is eligible.
 - If a floating panel that is currently closed matches the new display's category, that floating panel will be opened with the new display content. Floating panel use order will resolve which fixed panel to use if more than one is eligible.
 - If a floating panel that is currently open matches the new display's category, that floating panel's content will be replaced with the new display. If more than one open floating panel matches, the one with the oldest content will have it's contents replaced with the new display³⁸.
 - If a fixed panel matches the new display's category, the new display will replace the old content in that fixed panel. If more than fixed panel matches, the one with the oldest content will have it's contents replaced with the new display.
WGI – Maybe we should also have the option to let the operator select which panel a new display is going to open in.
- e) It is possible to configure panels to be the automatic "destination" for the workspace applications (e.g. diagnostics, batch operations, history view, etc.)
- f) One panel (either fixed or floating) in each framework is the "default destination" panel. When no specific panel is identified for new framework content and there is no category name match between the new content and any panel, new content appears on this default destination panel.

³⁶ At the cost of more window management responsibility.

³⁷ Users will tend to define display category names that reflect a combination of native display size and purpose.

³⁸ Supporting "round-robin" distribution of floating displays (of certain categories) to their corresponding floating panels.

- g) When an instance of the runtime workspace is first started (or goes through a "workspace hard reset") initial fixed panel contents are displayed. Each fixed panel may have an "initial content" (e.g. workspace display, or application) defined in the framework configuration. The configured initial content for fixed panel(s) may be overridden with information provided at runtime workspace startup time.
- h) Framework fixed panels may be configured to have several border options to help delineate them from other fixed panels in the framework. Options include "no border".
- i) A floating panel's upper left corner initially appears at a configurable "anchor point" position in the workspace framework.
WGI – Might be good to allow the association of floating panels to anchor points to be controlled by a scheme similar to the 'categories' which you have proposed above to associate displays to panels. So I could have maybe four anchor points with category 'FP'; any time I open a floating panel with category 'FP' it would select one of those matching anchor points according to an algorithm similar to that in d) above.
- j) Each floating panel is configured with a preferred size (height and width). It is also configured with a preference for how to choose the initial size:
 - Use native size: the native size of the display establishes the initial size of the floating panel window. (When the content has no "native size" (e.g. workspace applications) the floating panel's preferred size is used as the initial size. The floating panel display window will be constrained to not go beyond the workspace framework.
 - Use preferred size: the panel content is scaled (within any scaling constraints defined for the workspace) to fit within the preferred floating panel size, with possible gaps or scroll bars.
- k) Each floating panel is configured to be resizable or not. If the floating panel is configured to be resizable, it's window is rendered with a border suitable for use in window resizing operations. If the floating panel is not configured to be resizable, window resizing operations are not supported (the floating panel will always remain at it's initial size.)
- l) Some panels in a framework require a mechanism for user interactions with the panel. These include:
 - Closing the panel contents (e.g. floating panels)
 - Moving a floating panel
 - Picking the panel as the source or destination for a "copy contents to another panel" operation.
 - Panel content scaling or zoom operations.
 - Recall recent panel contents operations.
 - Open contents as new (Windows task bar) window³⁹.
 - Capturing a "snapshot" of the current contents.
 - Controlling the display of display layers.
 - Controls involving the "display using historical data" feature
 - Resizing floating panels (without dragging borders)
 - (other operations introduced in the future)

Also, depending on available space, users may want title or caption information about panel contents, for example:

- Display name
- Currently selected object (e.g. module, node) name

³⁹ When the workspace is in "another windows application" desktop management mode.

Framework panels may be configured to have "information and tool button" bars. The content of these bars is configurable. When display area is extremely limited, a mechanism is available to let the info/tool bar to be hidden or minimized.

WGI – this seems to call for an (optional) toolbar per panel, wherein the buttons would be limited in scope to actions that affect their own panel. May also need a 'workspace global' toolbar for actions that have broader scope.

4.5.2 Workspace Displays

Workspace displays are the mechanism to provide very customized content (and supporting display-related logic) within the runtime workspace. Workspace displays are intended to be used for:

- System specific "process graphic" displays; showing live process information and providing a means to manipulate process parameters.
- Logical "control panels" (or "faceplates") for DeltaV modules, nodes, etc.
- User customizable "pop-up dialogs"
- System status and alarm summary information
- Plant drawings or other documentation that may be annotated with live process information.

*(For details on what runtime workspace displays can be composed of, see the "**Starburn Process Display Primitives and Components**" concept document.)*

*(For details on the features and capabilities of display-related logic, see the "**Starburn Process Display Logic and Languages**" concept document.)*

4.5.2.1 User needs

- a) Workspace displays have names used for identifying them:
 - Users selecting them from display list/tree browsers, wanting to find/open a display by name.
 - Used by display elements (like display launch buttons) to open another display
 - Used by display logic to pick the appropriate display for the current conditionsRestrictions on display names should allow:
 - Them to be long enough to serve as their own description in a list of display names (allowing for localized names)
 - Characters that can server as delimiters (e.g. space, hyphen, underscore, etc.)
 - Upper/lower case preservation, but case-insensitive comparisons.
 - Easily reversible conversions to/from legitimate file system names
- b) We can anticipate users defining a large number of displays for their DeltaV system. To help provide organization of these for the operator, a hierarchical folder/directory system should be supported. Since folder/directory names in such a system will serve as their own description, the restrictions on these names should be comparable to those on display names.
- c) Some workspace displays (especially those creating by importing process instrumentation drawings from non-DeltaV applications) are likely to contain display elements⁴⁰ that are not always appropriate for display under typical conditions. Yet, for special conditions (or when used by engineering or maintenance personnel) getting access to this information would be valuable.

Workspace displays will support display elements associated with "display layers"⁴¹. The display configurer can segregate information into separate layers, and choose which layers are visible by default when workspace displays are first rendered in the runtime workspace.

⁴⁰ Information appropriate to the original drawing, but has little to do with the DeltaV system and the automatic process controls; e.g. piping tags, size and materials, manually operated valves, visual inspection points, etc.

⁴¹ Similar in concept to the "drawing layers" features in CAD applications.

The runtime workspace will provide a mechanism, at the framework panel level, to indicate what layers are available in a display, and change which layers currently appear. For each layer, three levels of visibility is supported:

- Hidden: display elements associated with that layer are not rendered.
 - Subdued: display elements associated with that layer are "ghosted" (configured level of transparency) behind any full view layers.
 - Full view: display elements are rendered with their normal (configured) color and transparency level (usually opaque)
- d) Workspace displays may be configured to offer special display layers automatically created by the runtime workspace. Special display layers provide an mechanism to add "smart features" to user configured displays with little extra display configuration work. Special display layers that should be considered include:
- Auto "module controls": tiny module "faceplates" appear on top of display elements with data links to module parameters. (Automatically "weeding down" to at most one faceplate per module.)
 - Auto "trend views": small, semi-transparent real-time trend windows appear next to display elements involving numeric data links. Trend windows support interactions to quickly delete unwanted ones, drag to better position, increase size, make opaque, or launch the a full function process history view application in the appropriate panel.
 - Auto "alarm volume history": for each plant area and Unit associated with data links on the display, an active alarms by priority vs. time column chart is displayed; indicating when and where alarm bursts occurred.
- e) While the "fixed and controlled" nature of the runtime workspace framework is intended to relieve operators of most window management activity, for some users it may be too rigid. Resolving and monitoring different process problems may call for more flexibility in using the available display space.

Specialized "sub-panel" workspace displays may be configured. Though capable of containing typical display elements, they are distinct in that they contain panels that support the configuration and behaviors of framework fixed panels.

Sub-panel displays offer a couple of features to help users better tailor the runtime workspace to the problems of the moment:

- By choosing a sub-panel display, the operator can dynamically subdivide one of the panels in the framework into a pre-defined arrangement of smaller fixed panels. (Consider sub-panel display intended to subdivide the "main" display panel, into a row of "faceplate" sized panels on top, and a wide panel ideal for a trend window on the bottom.)
- Once the panel contents have been defined in a sub-panel display, the operator can save the display (in temporary user-session display storage) for later use in the session. Essentially, instances of new displays, dynamically composed to address specific process monitor problems, can be created and reused by that user/workstation/session.

4.5.3 Workspace applications

Workspace applications are the means to deliver task-specific, optimized, "ready out-of-the-box" user interfaces via the runtime workspace. These user interfaces require little or no engineering or customization between DeltaV systems. They also offer an opportunity for richer user interfaces; ones that would be difficult to accomplish through combining display primitives and components and display logic.

Workspace applications differ from normal Windows applications in that:

- They communicate with and their lifetime is controlled by the runtime workspace.

- Expecting not to be "full screen" applications, they take particular care to automatically adjust their user interface layout to best fit the panel area they are assigned to work within. When used in a floating panel, they make every attempt to provide a suitable layout within the initial preferred size configured for the floating panel. As the floating panel is resized, they adjust their layout accordingly.
- They follow the runtime workspace user interface conventions for common actions (e.g. selecting, scrolling, scaling, etc.) so the entire workspace feels well integrated.
- They follow workspace conventions for retaining user interface state, so that "go back to last display" operations for panels work well.
- Where possible they offer useful "tracking" modes of operation; where their content/focus automatically follows the system object (module, node, FF device, batch, logic solver, etc.) that has focus in other panels in the framework (usually the "main" or "default destination" panel). They offer a means to disable tracking mode.

We would expect that adding new workspace applications would be used frequently as the capabilities of the DeltaV system increases.

5 Application Architecture

5.1 Background

From the beginning of the Starburn project it was acknowledged that we would be addressing several objectives. Although these objectives are a subset of what is presented at the beginning of this document, it is worth repeating what we understood at the beginning of the project.

1. Decouple Clients from Servers
2. Improve Overall Configuration Integration
 - A common framework
 - Common controls
 - Large System configuration performance improvements
3. Support Multiple Development Centers
 - Austin
 - Leicester
 - Philippines
 - Pittsburg
4. Separate Services and Functional Development
 - Build and release services and applications separately
 - Run builds independently at multiple sites
 - SS Management would be separated as much as possible
 - Integration of Code-sets on a weekly basis
 - Only ship core DeltaV infrastructure build + assemblies to remote sites, source code to be shipped as required
5. Support Multiple DCS's
 - DeltaV
 - Ovation
6. Address new functionality
 - Operator Workstation Replacement
 - Process Graphics based on Scalable Graphics Technologies
 - Process Modules
 - Abnormal Situation Prevention

With these objectives in-mind we first tackled the task of building configuration applications (Operator Workstation Applications, Process Graphics, Process Modules, and Abnormal Situation Prevention all have configuration aspects to them).

We started the effort by breaking the problem down into client-side and server-side models. The configuration application architecture is presented first followed by a more general model.

5.2 Configuration Applications

5.2.1 Server-side Configuration Model

The server-side configuration model is shown in the following drawing.

Server Process

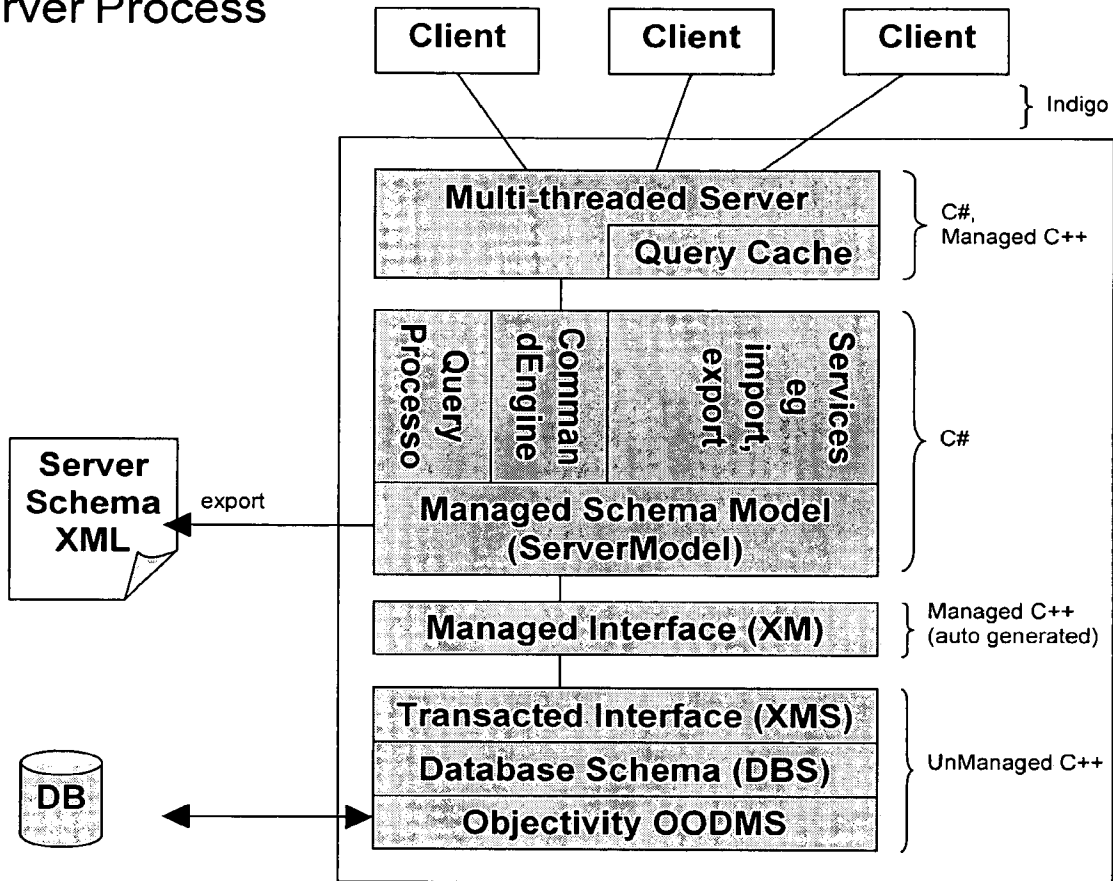


Figure 34. Server-side Configuration Model

At the lowest level of the model are the UnManaged C++ components – **Objectivity**, **Database Schema**, and **Transaction Interface**. These layers have been with DeltaV from the beginning and will continue to be extended and utilized as new functionality is added to DeltaV. Since the Starburn development is being developed as managed code using the .Net Framework and C#, it was necessary to put an interop layer into place. The **Managed Interface (XM)** provides this function. The rest of the functions in the model address the service interface requirements. These are discussed below.

5.2.1.1 Managed Schema Model (ServerModel)

For a service based model to be versionable and maintainable long-term, it is necessary for the internal schema to be published. In this layer the entire model for DeltaV configuration is exported into **Server Schema XML**. The model for doing this is illustrated in the following drawing:

An example of server-side Object Model is shown below.

Published Server Model

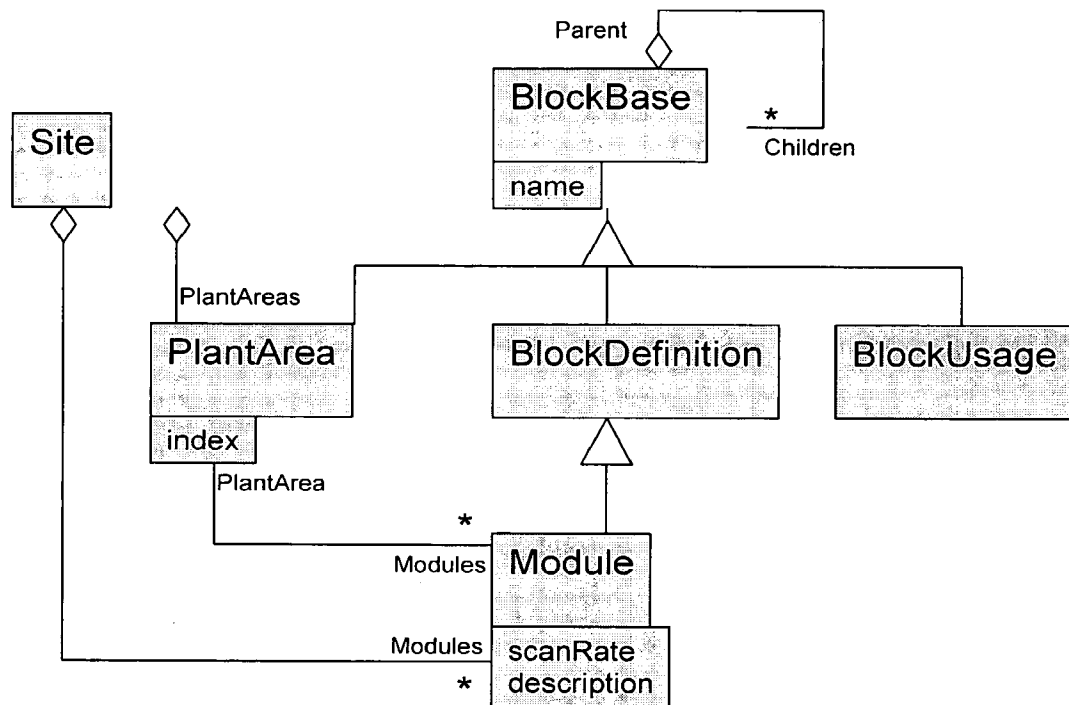


Figure 35. Published Server Model

Clients need access to the Server Schema. Since loading the entire schema is impractical for many reasons, another mechanism is needed. The mechanism is described in the Query and Commands sections which are described next.

5.2.1.2 Query Processor

The biggest change for clients is the introduction of the Query Framework. As was described in the previous sections, the Query processor is written in C# and coded against the XM layer which in turn is a thin veneer C++ layer interface between the managed world of .NET and the unmanaged XM classes of HawkDB.

These query objects are very dynamic. A client asks for a query to be executed and gets the result as XML. The client request identifies the .NET assembly and query. The framework loads the assembly if necessary and executes the query. This provides good decoupling of development paths.

The Query engine is shown below.

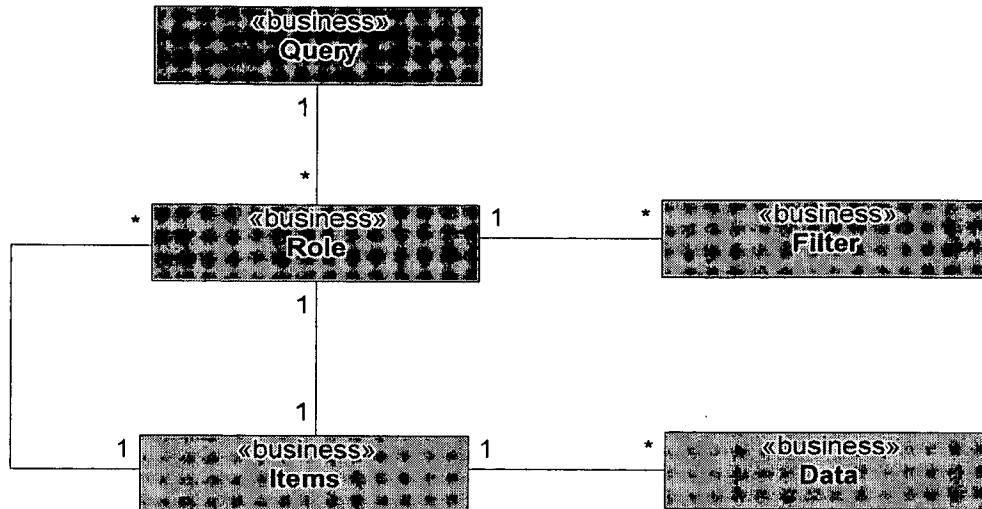


Figure 36. Query Model

An example Query and it's returned result is shown below:

Root *Role* *Filters* *Role* *Property Requests* *Role*
Site.PlantAreas[name='AREA_A'].Modules(description, scanRate).blocks

```

<Module name="MOD1">
  <Properties>
    <description>control module</description>
    <scanRate>1</scanRate>
  </Properties>
  <Role name="blocks">
    <Block name="usage1"/>
    <Block name="usage2"/>
  </Role>
</Module>
<Module name="MOD2">
  ...
</Module>

```

The Query semantics allow the user to specify whether the request is to register with the database for updates.

We store xml chunks in the **Query Cache**. These chunks are loosely coupled by nature. The query objects created to load and save these xml chunks do nothing more. There's no business logic here. Now we produce/cache exactly the kind of data we want to transport around the system, ie XML. It's a flexible architecture that's extendable and can be molded by the client application writer to suit their needs.

5.2.1.3 Command Engine

The commands interface is used to command the database server to do things – for example to save a module, create a module, etc. An example command is shown below:

```
Import System
Import DeltaV.Config.Command
Import.DeltaV.Config.Model

Function createModule ( areaName, moduleName )
{
    var area = Site.role("PlantAreas").key(areaName) ;
    var mod = area.role("Modules").Create(moduleName) ;
}
```

5.2.1.4 Services – Import/Export, etc

There are several functions supported by the database server that are best managed as individual services. These services include Import, Export, Download, Upload, Commission, Commission Fieldbus, etc.

5.2.1.5 Multi-threaded Server

The multi-threaded server code is designed to support many applications. For queries and commands a thread will be allocated for the duration of the command / query. For service such as import / export a thread will be allocated for the duration of the import / export.

5.2.2 Client-side Configuration Model

Server-side support is half of the solution. Clients need to be restructured to utilize the query / commands style of interface. The client-side configuration model is shown below. Generalizing this model leads to the Application Architecture – the subject of this chapter. Before describing the general model the key blocks in the Client-side configuration model will be discussed.

Client Process

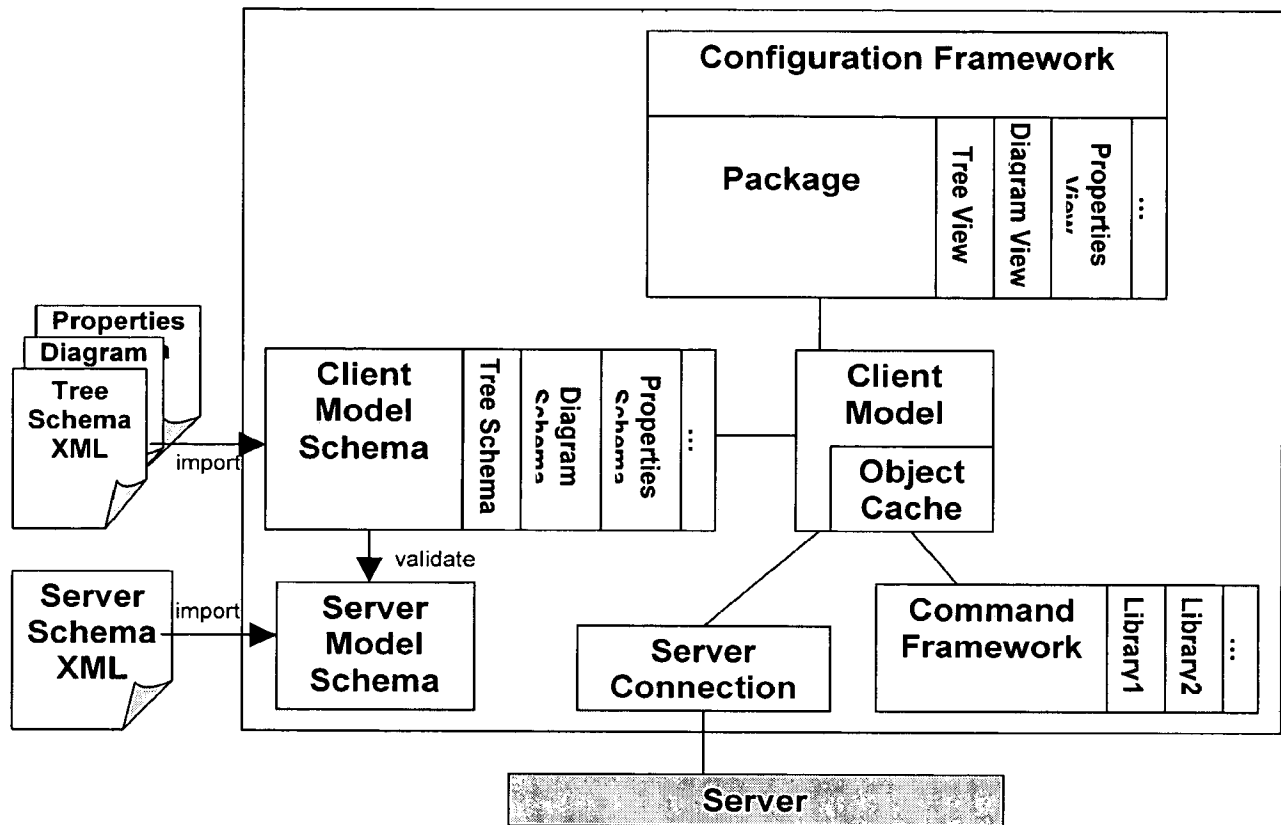


Figure 37. Client-side Configuration Model

This drawing does not show the `IDataItem` and `IDataSource` layer that clients can take advantage of. These are described under the section Application Framework below.

5.2.3 Client Model Schema

Each configuration client is made up of many views that work together to allow the user to add structure, change properties, etc. Each view in a client has very specific display requirements. Although the data model behind the configuration clients is based on the **Server Schema XML**, the data model is quite often not exactly the same (configuration clients often have additional items that the server schema has no knowledge about). In addition, each view in a client is normally interested in a small sub-set of the **Server Schema**. So how do we bridge these differences? The solution is to provide a **Client Model Schema**. As is shown above, it is possible (but not necessary) to customize the client model schema for Tree Views, Diagram Views, Properties Views, etc. An example Client-side Schema is shown below.

```
<Type name="TestTypeA" serverType="ServerTestTypeA">
  <Properties allServerProperties="F">
    <Property name="name" dataType="string" serverPropertyMapping="(name)" />
    <Property serverPropertyMapping="(description)" />
    <Property serverPropertyMapping="ServerRole1[type='ServerType2'](subProp1)" />
  </Properties>
</Type>
```

```

    <Property serverPropertyMapping="ServerRole2[type='ServerType3'](subProp2)" />
    <Property name="calcProp" dataType="string" command="GetProp" assembly="commands" />
  </Properties>
  <Roles allServerRoles="F">
    <Role name="ClientRole1" serverRoleMapping="ServerRole3" />
    <Role name="ClientRole2">
      <ServerRole mapping="ServerRole4" />
      <ServerRole mapping="ServerRole5" />
    </Role>
    <Role name="ClientRole3" serverRoleMapping="ServerRole6[type='ServerType6']" />
    <Role name="ClientRole4" serverRoleMapping="ServerRole6[type='ServerType7']" />
  </Roles>
</Type>

```

Making use of Roles and Types to map from Server-side to Client-side is illustrated in the following diagram.

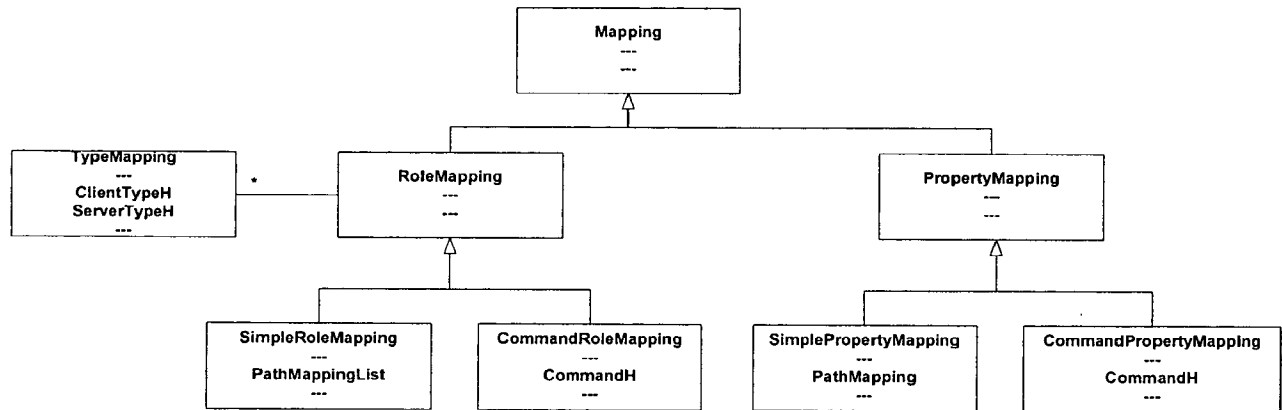


Figure 38. Mapping Server-side model to Client-side model

5.2.4 Client Model and Object Cache

As the client loads Server-side schema and processes it using its own Client-side schema and logic, the client model and object cache are loaded up. An illustration of this is shown in the following diagram:

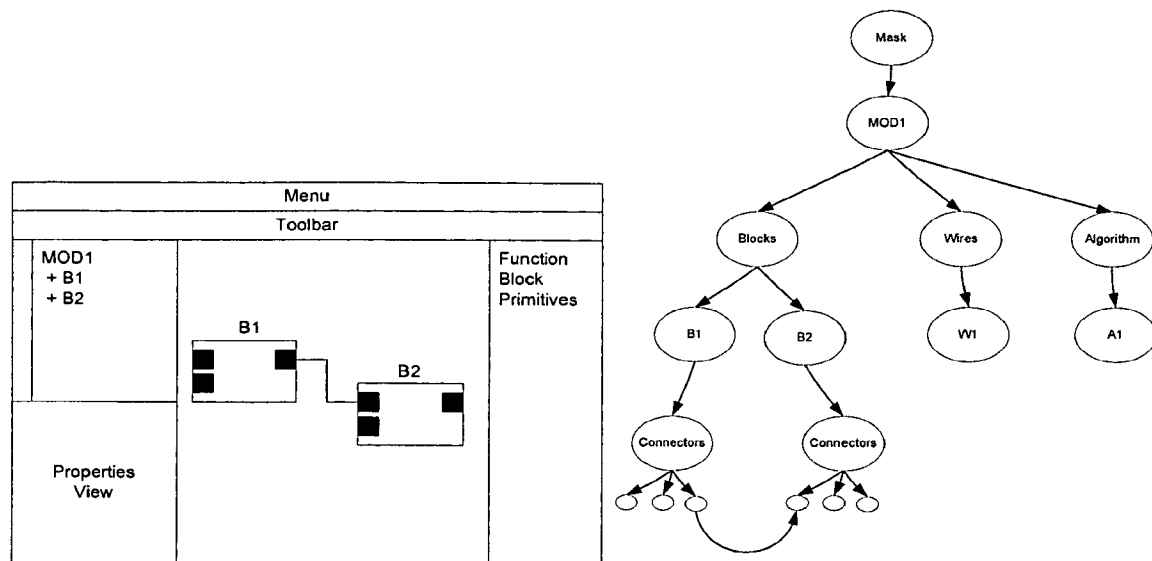


Figure 39. Client Model

5.2.5 Command Framework

The commands framework is used to create commands that are passed from the client to the server. Examples of commands include "Import ModA", "GetBitMapCmd()", etc. The example below illustrates a client looking up a command to get a bitmap.

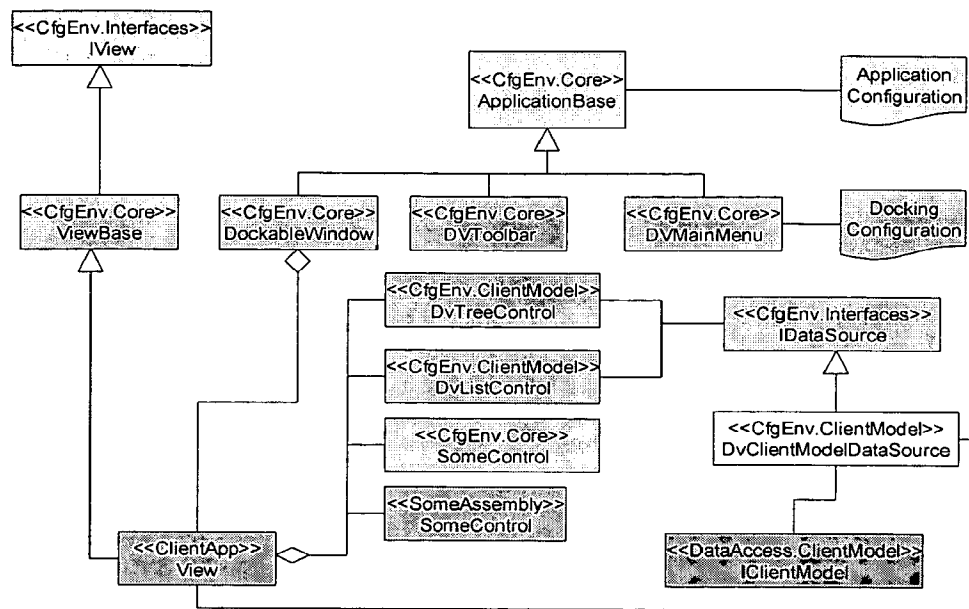


Figure 41. Configuration Framework

Details on the Configuration Framework are provided below.

5.2.7 Package

The **Package** contains the functional logic for the application. For example, if this was Control Studio, this would be the Control Studio configuration logic.

5.2.8 Views

Applications are made up of many cooperating views. These views work together through the package and client model to perform the actions as requested by the users. Views will generally be based on a common control such as a **Diagram Control**, **Tree Control**, **Properties Control**, or **Grid Control**.

5.3 Functional Breakdown of Applications

The application architecture is a set of frameworks, patterns, and styles to properly use the Technical Architecture and build Starburn configuration, operator, control, etc interfaces. The application architecture needs to be flexible enough to deal with very large scale applications (e.g. our configuration applications with 1000's of classes), heavy use and strong performance requirements (especially our operator interface applications where more than a 1 sec response is considered unacceptable), and data consistency and integrity requirements (in the case of configuration a supporting transaction model is required, in the case of runtime confirmation or acknowledgement is required – user changes cannot get dropped on the floor).

The general model for applications is illustrated below:

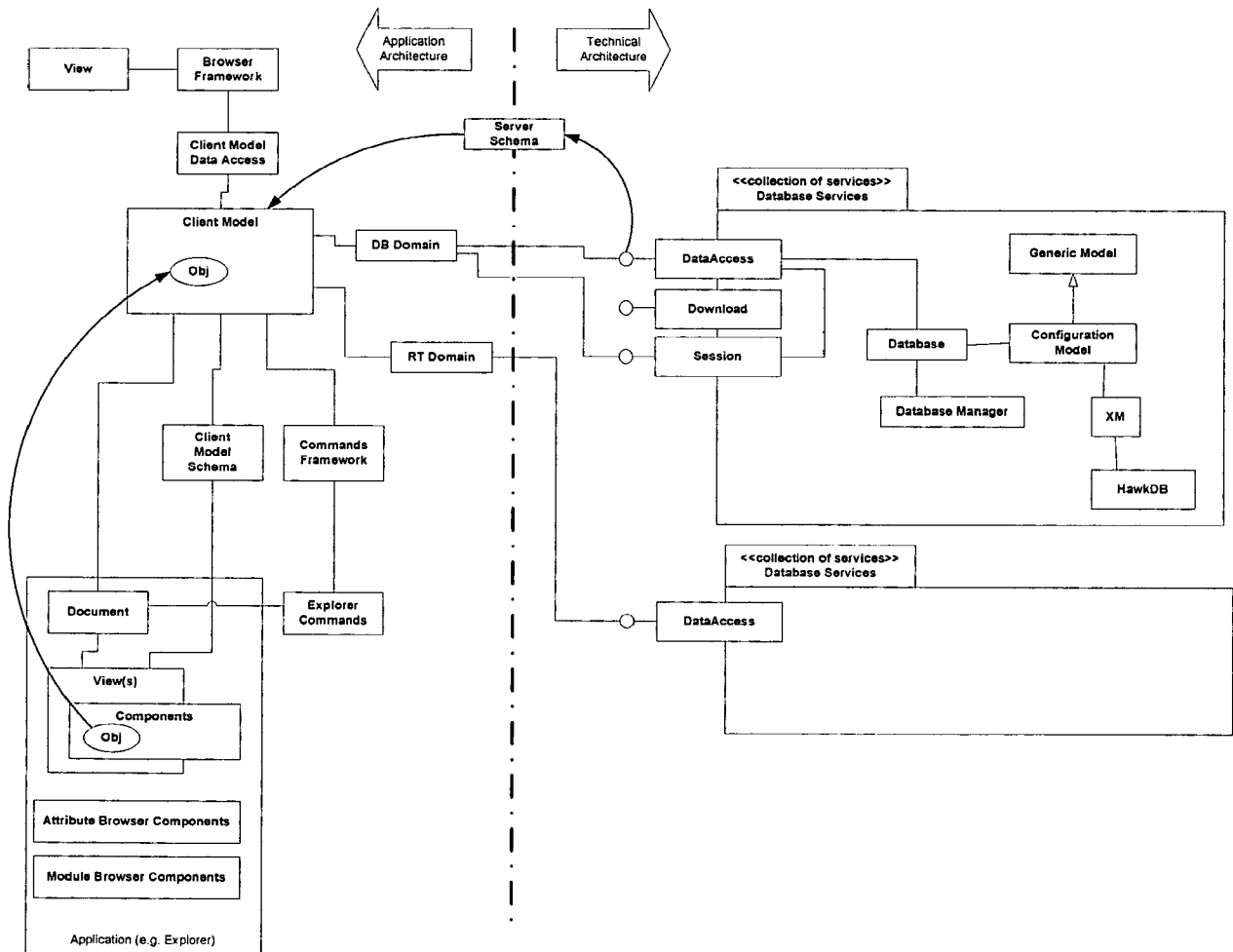


Figure 42. Application Architecture – Shown with DeltaV Explorer

There are many aspects to what is shown in the drawing above. Everything to the right is described in the Technical Architecture. Everything to the left is part of the Application Architecture.

DeltaV applications themselves should be viewed as a set of isolated reusable components. Components will be developed independently of each other and the applications that use them. Any individual component can make use of core services or other shared interfaces (for example the client model). They may also provide plug-in points for extensibility.

Functional components can be tested in isolation and developed independently of each other and the applications that use them. Development of these components can be moved to other geographical areas.

5.3.1 Controls

Many of the views in applications repeat across many applications. For example the Diagram Control is used in all of the Module Editors, Process Graphics, as well as other applications. A partial list of controls includes:

- 1- Diagram Control
- 2- Properties Control
- 3- Tree Control
- 4- Grid Control
- 5- Structured Text Control

5.3.2 Views

Many of the views in applications are repeated across applications. For example, the parameters dialog should be the same across all DeltaV configuration applications. A list of common dialogs includes:

- 1- About dialog : The About dialog and property pages are on all applications.
- 2- Fieldbus dialogs : There are many Fieldbus dialogs for showing device and resource information. These dialogs need to be the same regardless of which editor is invoked.
- 3- File dialog : The file dialog is customized for DeltaV – in the case of the Database it's to open database things.
- 4- Hart dialogs : There are many Hart dialogs for showing device. These dialogs need to be the same regardless of which editor is invoked.
- 5- License dialogs : Licensing dialogs are invoked from many applications – in some cases from several locations within the same application.
- 6- Parameter and Property pages : Property dialogs are shareable components which are usable from any application. Each property dialog is tied to a particular configuration data type.
- 7- User preferences : User preferences are set on applications – they should work the same everywhere.
- 8- Wait and Cancel dialogs : From time-to-time applications take extra time to complete. When this happens Wait and Cancel dialogs need to be invoked.

5.3.3 Browser

There is a single reusable UI browser component which can be used to provide the UI to multiple separate item browsers. There are multiple reusable browser models, each of which determines the behavior of a browser for a particular type of item. There are multiple browser data source adapters allowing the browser to browse into multiple data sources.

An application brings together a browse data source with a reusable browser model and the reusable browser UI in order to browse for a particular class of item.

5.3.4 Download

There is a reusable UI component which can drive the server download process. The download component can be called by any application, passing the server data handles to items to be downloaded. The download component will provide user feedback and sequencing for the download process.

The UI download component makes use of the download core service.

5.3.5 Drag/drop

There is a common drag/drop format used by all DB operations. There is a UI command which will call to the server config model to determine whether drop is allowed and to action the drop if it occurs. The configuration model in the database server understands enough of the semantics to perform the correct action when a paste occurs.

5.3.6 Cut/copy/paste

There is a common paste buffer format used by all DB operations. There is a UI command which will call to a database data access function to determine whether paste is allowed and to action the paste if it occurs. The configuration model in the database server understands enough of the semantics to perform the correct action when a paste occurs.

5.3.7 Other UI commands

User interface commands are shareable components which are potentially usable from any application. Each command contains the user interface and knowledge of client schema commands necessary to perform a particular action in a particular context.

5.3.8 Application

Applications build on the shared components identified above. Examples of DeltaV applications are

- Explorer
- Control Studio
- Diagram Editor
- Process Module Editor
- Operator Graphics
- Expert Systems

Applications are developed independently in the same way other components are developed.

5.4 Configuration Framework

5.4.1 Overview

The Configuration Framework is being designed to support client applications that use different data stores – sometimes more than one in the same application. With this in mind it's clear that the configuration framework cannot require the client developers to have a dependency to the runtime or the client model. The framework will be implemented in a way that allows application developer to pick and choose pieces that are needed depending for an application. For example developers will be able to use the docking framework with out being forced to use the command management or user preferences.

The configuration framework will have several layers that build upon each other. These layers are illustrated in the diagram below and then followed with an explanation.

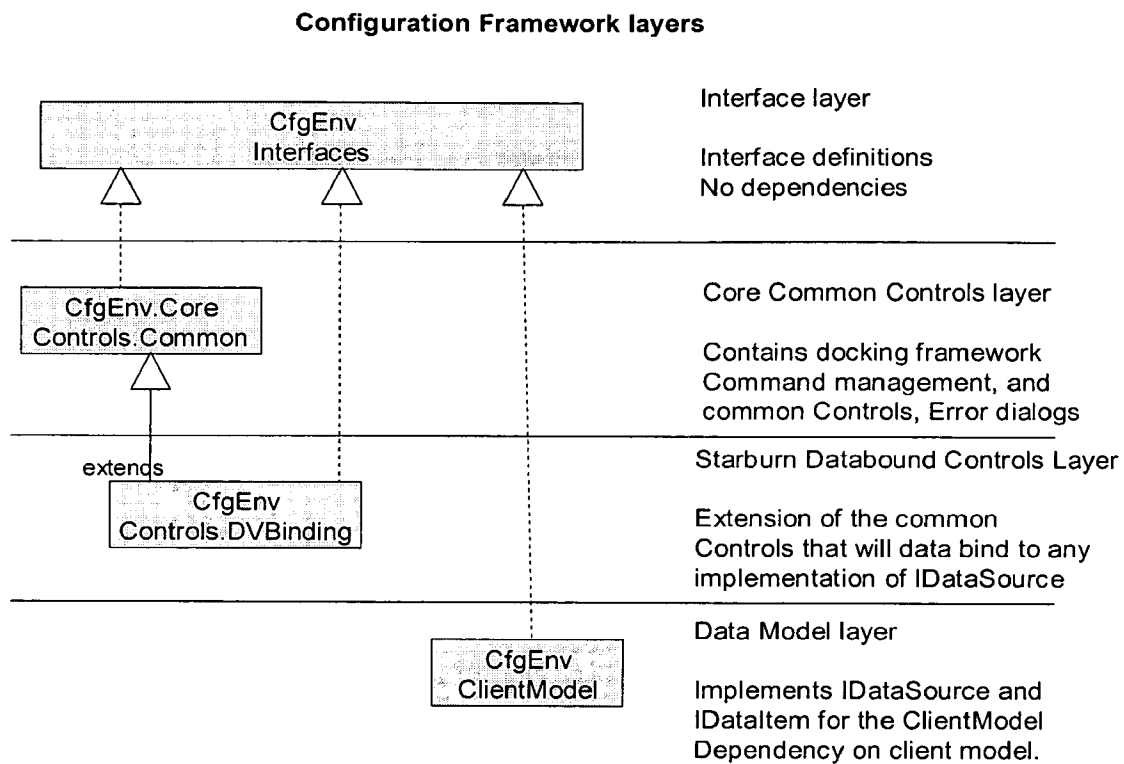


Figure 43 Configuration Framework Layering

The first level will define the interfaces used by all subsequent levels. Next is the level that contains all the common controls and extension of .NET controls. In some cases such as a tree control, this may be little more than providing defaults to certain properties so all applications have a similar look and behavior. Other controls will be composite controls such as a palette control or new controls that do not exist in .NET such as an IntelliSense control, diagram control, and frameworks for docking and command management.

The next level will extend the controls in the previous level's controls that utilized several in house interfaces to allow them to bind to object models that provide implementations of the interfaces. These are the interfaces such as IDataSource and IDataItem. Although this level will be able to bind to Starburn data models this level does not have a dependency to the client model or runtime.

A default implementation of the binding interfaces using the client model will be provided since many configuration applications will use the client model. This implementation can be used directly or extended as necessary by client application developers. This portion of the framework will have a direct dependency to the client model. Clients that use the databound portion of the framework will have at least an indirect dependency on the client model.

It is possible for client applications or other components to use the framework in any of the following scenarios or combinations:

- (1) Interface layer - The interface layer is the glue between controls and the data models, allowing the UI portions of the framework to be independent of the data models. Developers may implement or use one or more interfaces to allow the framework to interact with the data or application.
- (2) Common.Controls layer – The client application uses common controls provided by the Configuration Framework. There is no other dependency on DeltaV or the data that is rendered in the controls. For instance, a client could populate the data manually or use .NET data binding to data bind the control to a SQL database or XML file. The client application would inherit the look, feel and behavior of the common controls in their application. This layer will contain features such as docking, command management and controls.
- (3) Starburn Databound Controls layer - The client application uses controls provided by the Configuration Framework IDataSource layer. There is a dependency on the control and the data rendered in that control. The dependency is that the data rendered in the controls uses IDataSource and IDataItem.
- (4) DataModel layer – The client application uses data from the ClientModel. The client application could write their own controls to display the data or use the IDataSource controls provided by the framework.

Developers will be able to use portions of the framework without the requirement to use everything. For instance the docking will not depend upon the command management. Common controls can be used with or without the Starburn data binding. Additionally, the controls can be used without requiring the data binding interfaces to be implemented.

5.4.2 Assembly Dependencies

The Configuration framework will be divided into several namespaces. The assemblies will be divided to allow development of client application that do not have a dependency, even indirect, to the client model or runtime. All client model specific implementations will be in the DeltaV.CfgEnv.ClientModel assembly. If necessary a DeltaV.CfgEnv.RuntimeModel assembly will be created that has a runtime dependency.

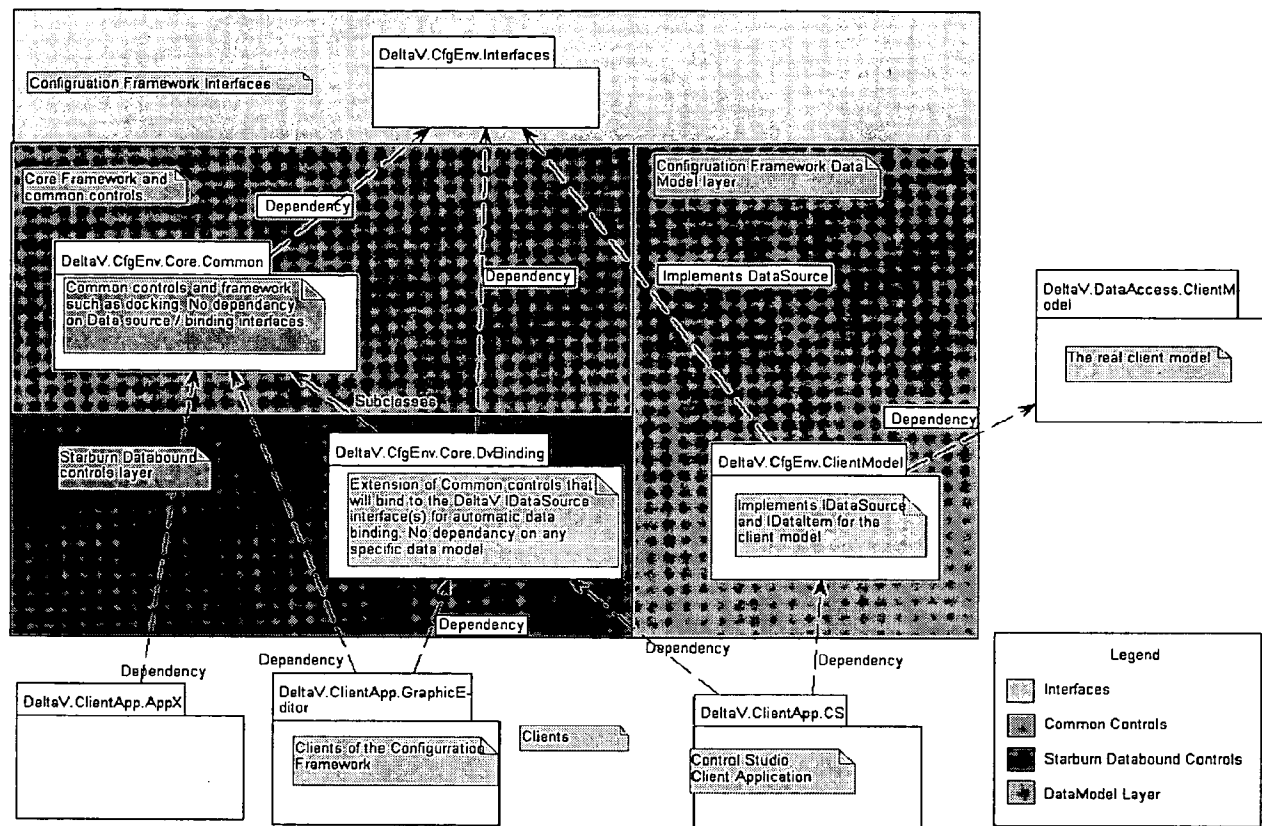


Figure 44 Package Dependencies

DeltaV.CfgEnv.Interface: This interface assembly contains all interfaces implemented by the Core package, used by the clients as well as any interfaces that the client must implement.

DeltaV.CfgEnv.Controls.Common: This assembly implements common controls and extends .NET Framework controls to conform the UI Guidelines. The class can be used directly or indirectly in client applications. Examples are tree controls, palette control, intellisense controls. At this level the controls do not implement in-house data binding. This assembly and these classes do not have any Starburn or DeltaV specific dependencies. This assembly can be used directly by a client application and will not create a direct or indirect dependency to the runtime or client model.

DeltaV.CfgEnv.Controls.DvBinding: This assembly implements extends the common controls in the Core.Common assembly and uses the in-house data binding interfaces, `IDataSource`, to automatically load, update and send events load. These classes do not have specific any specific Starburn or DeltaV dependencies. This assembly can be used by client applications and will not create a director or indirect dependency to the runtime or client model.

DeltaV.CfgEnv.ClientModel: The configuration client model package will provided a default or base implementation of the Starburn Configuration framework defined data binding interfaces that interact with the client model. This assembly will provide additional framework for those client applications that use the client model as their data source creating a dependency to the client model.

5.5 Browser Framework

The Starburn Browser will consist of a Browser framework, Browser View, individual Browser DataSources and packages for different Browser Types. The Browser View will query the Browser Framework for specific information to display. It is possible to develop Windows based or Web based Browser View, using the same underline business logic in the Browser Framework.

The Browser Framework defines the contracts between the Browser and individual Browser Data Sources. The Browser Framework communicates to Browser DataSources to extract information for the Browser View via the interfaces defined in the contracts. The realization of these interfaces lies in each data source.

For each application that utilizes the Browser, there will be a set of packages to facilitate different browser types within that application. For instance, there will be a ModuleBrowser package for the module browser of DeltaV Explorer, where the context menu of the Module Browser will be constructed by the objects within the ModuleBrowser package. The execution of the context menus will be captured in the specific Browser DataSource. Code re-factorization will take place in the future to maximize code-sharing.

There will be a Browser Framework that contains the contracts (interfaces) between the browser and individual data source.

Here is a brief look of the project layout:

```
\Starburn
  \Core
    \Mainline
      \BrowserFramework
        \Browser
        \BrowserDataSource
    ...
  \BrowserView
    \Mainline
  ...
  \ClientModel
    \Mainline
      \Browser
      \ClientModelBrowserDataSource
  ...
  \DeltaVExplorer
    \Mainline
      \Browser
      \Packages
        \AttributeBrowser
        \ModuleBrowser
      ...
  ...
  \ProcessGraphicsDataSource
    \Mainline
      \Browser
      \ProcessGraphicsBrowserDataSource
  ...
  \ProcessGraphicsEditor
    \Mainline
      \Browser
      \Packages
        \PumpBrowser
        \ValveBrowser
```

To facilitate multi-version development, the current plan for output of the projects is:

```
\bin
\DeltaV.ClientModel
  clientmodel.dll
\DeltaV.BrowserFramework
  browserframework.dll
\DeltaV.ClientModel.Browser
  clientmodelbrowserdatasource.dll
\DeltaV.Explorer
  exp.exe
\DeltaV.Explorer.Packages
  modulebrowser.dll
  attributebrowser.dll
```

6 Technical Architecture

6.1 Background

The Starburn project extends the DeltaV core using a service-oriented approach. Service-oriented differs from Object-oriented primarily in a several ways:

- 1- Service-oriented architecture emphasizes loose coupling vs. Object-oriented which emphasizes reuse.
- 2- Service-oriented architecture share schema and contract vs. Object-oriented which share classes.
- 3- Service-oriented architecture views an application as a system that is built from autonomous services vs. Object-oriented which views an application as a program that is built from class libraries.
- 4- Service-oriented architecture views communication as expensive and explicit vs. Object-oriented which views communications as cheap and implicit (e.g. DCOM's marshal by value approach).
- 5- Service-oriented architecture views deployment as an on-going process vs. Object-oriented which views deployment as a unit.

The notion that communication is expensive not only applies to inter-service communication which may be spread out across several nodes or potentially over a wide-area network, but to developer communication. Even in scenarios where services run on one node or at most on a few nodes connected together across a LAN, developers may be distributed across geographical and organizational boundaries. Each type of boundaries increases the cost of communication. Service-oriented architectures adapt to this model by reducing the number and complexity of abstractions that must be shared across service boundaries. The number of interfaces and the number of methods available which each interface is kept as small as possible.

During the 1990's Object-oriented programs such as DeltaV made huge strides in carefully constructing class hierarchies' emphasizing reuse. A side-effect of this however was close coupling. Although it was possible to build applications to be independently deployed this was not the normal case. A Service-oriented approach addresses this problem differently. Duplication is considered more favorable than coupling.

Service-oriented architecture views deploying applications as an ongoing process. It is expected that applications will be versioned independently from core services. While services themselves will continue to be distributed as a whole, the aggregate state of the overall system and applications will be mixed. It will be common for services to be deployed long before applications are available to make use of them. Services and applications are also expected to evolve over time. The degree to which services can be updated and new services introduced depends on the complexity of the service interaction and the ubiquity of the services, i.e. what has to be present at the same time in order for a feature or application to function correctly. The great risk with services is that "assumptions" will be made about their use which in turn become expected behavior over time. It is important to keep the interfaces as simple as possible so that both services and applications can be versioned with these "assumed" side-effects.

Services share schema and contract not classes. In the Object-oriented paradigm developers create classes which in-turn share both structure and behavior in a single named unit. In the Service-oriented paradigm services interact based on schemas (for structure) and contracts (for behavior). Every service publishes a contract that describes the structure of the messages it can send and receive. The ordering of the messages must be carefully document. Contracts and schema often make use of XML.

There are downsides to this decoupling as well. Unlike traditional class libraries a service must be careful to validate input data on each message. Once defined the names and signatures of services must remain constant across all space and time.

6.2 DeltaV Concept for Service-oriented Architecture

As described in the previous section entitled "Background on Service-oriented Architecture", the DeltaV services concept is to divide the system into loosely couple segments. Each segment has a clearly defined external interface that can be tested and versioned. The entire DeltaV system is made up of segments that build on other segments. Each segment can be tested in isolation through its interface.

Two kinds of segment are identified. Core DeltaV functions are presented as services and user applications are broken down into functional groupings.

Core DeltaV functions, such as configuration database access, runtime access, history access, are implemented as services which are available to client applications through service interfaces. These service interfaces are designed to be generic in nature and their signature to be entirely independent of the schema of data in the configuration or runtime database. These service interfaces should therefore need to be changed if new service capabilities need to be added, not if new data objects are added to the DeltaV system.

The services identified for DeltaV consist of the following:

- 1- Discovery Service
- 2- Session Service
- 3- Runtime Services
- 4- Database Services
- 5- Versioning Services
- 6- Historian Services
- 7- Historian Scanner Service
- 8- Alarms and Events Services
- 9- OPC Data Services
- 10- XML File Services
- 11- HTTP Links Services

Each of these services will have a very specific interface defined for it. System Services implementing these services will follow the service interfaces. This allows clients to remain constant even when services are swapped out. For example, as part of the DeltaV and Ovation installations, specific installations of the Service Providers will be installed. This is illustrated in the drawing below.

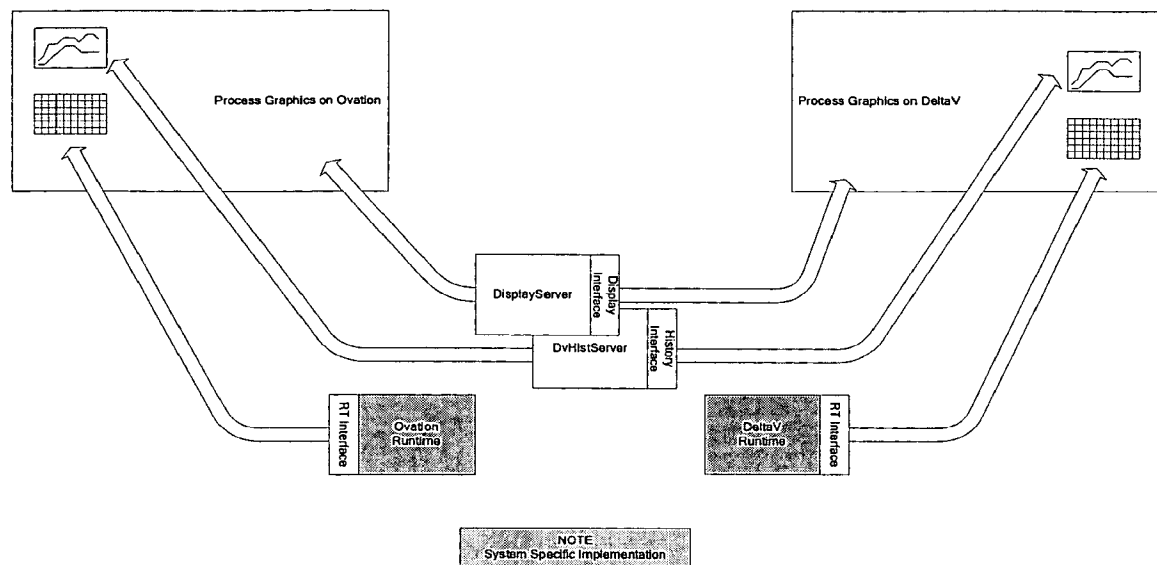


Figure 45. Implementing System-specific Services

The following enumerates the expected service interfaces which will be provided and the functions to be available on each. This section is meant to serve as a brief introduction to the services. A more in-depth discussion of the services is provided in the following sections.

6.3 Services

6.3.1 Discovery

The discovery service provides two key functions – it provides support for registering services on each node and it provides a general query mechanism for locating services locally as well as anywhere in the system.

Services are located by their URI. Examples of the URI are shown below:

Identity Role

Soap://www.contoso.com/pppp/whatsnew.aspx?date=today

					-optional query string
					-path identifier
					-port
					-server identifier
					-

Transport address for http

```
|      |      |      |      | -optional query string
|      |      |      |      | -path identifier
|      |      |      |      | -port
|      |      |      |      | -server identifier
|      |      |      |      | -scheme identifier
|-
```

We have our own scheme identifier.

```
|      |      |      |      |-optional query string
|      |      |      |      |-path identifier (our service names)
|      |      |      |      |-port
|      |      |      |      |-server identifier
|      |      |      |      |-scheme identifier
|-
```

```

sequenceDiagram
    participant Host as <<Service Host>>  
Discovery Service
    participant Server as <<Service Host>>  
Runtime Server

    Host->>Server: 
    activate Server
    Server->>Host: 
    deactivate Server

```

The diagram illustrates the interaction between two service hosts. On the left, the **<<Service Host>> Discovery Service** contains a **<<service>> Discovery** component with the following methods: `registerService`, `locateService`, and `publishService`. On the right, the **<<Service Host>> Runtime Server** contains a **<<service>> DataAccess** component with the following methods: `Open(SessionH, callback)`, `Close`, `AddGroup(GroupH, *GroupH)`, `AddItem(GroupH, *Item, *ItemH)`, `ReadItem(ItemH)`, `WriteItem(ItemH)`, `SyncRead(ItemH)`, `SyncWrite(ItemH)`, `RemoveGroup(GroupH)`, and `Broadcast("Locate", "ServiceName")`. A dashed arrow points from the **Discovery** service in the Discovery Service host to the **DataAccess** service in the Runtime Server host, indicating a dependency or interaction.

107

These services are described below. Scenarios are then presented illustrating how the services are used.

6.3.1.1 Service Descriptions

6.3.1.1.1 Data Access

The data access service provides an interface to allow querying for information about the services that are installed on each node. If the service is not located on a specific node there is mechanism for locating the service in the system.

6.3.1.1.2 Discovery Service

The discovery service provides an interface for registering services on each node.

6.3.1.2 Scenarios

6.3.1.2.1 Register Service

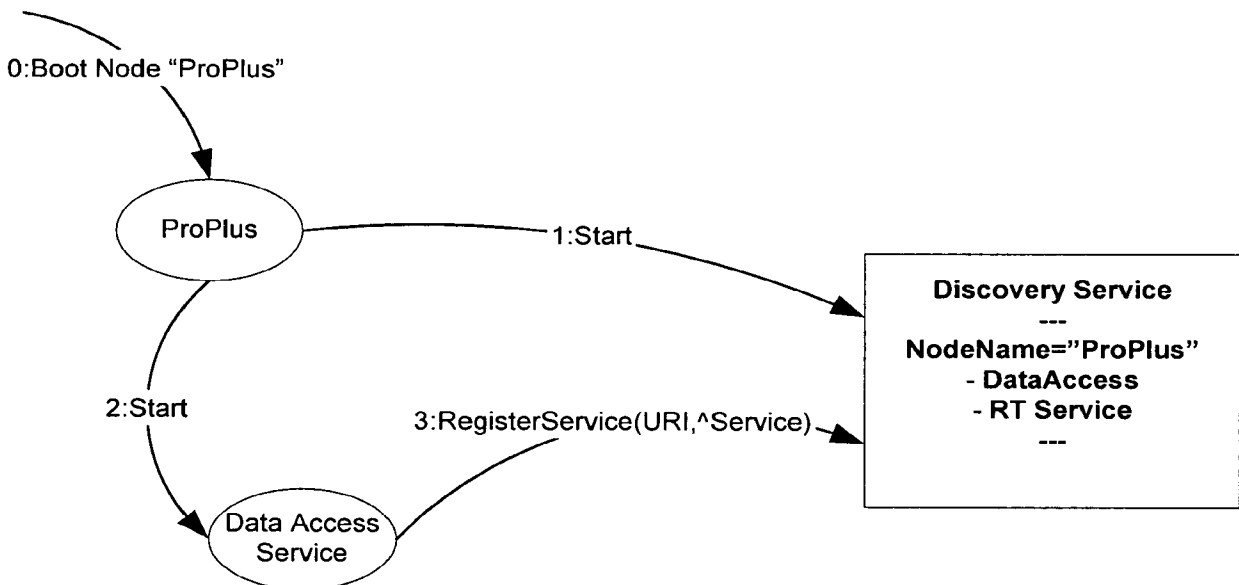


Figure 47. Discovery Service - Register Service

6.3.1.2.2 Locate Service

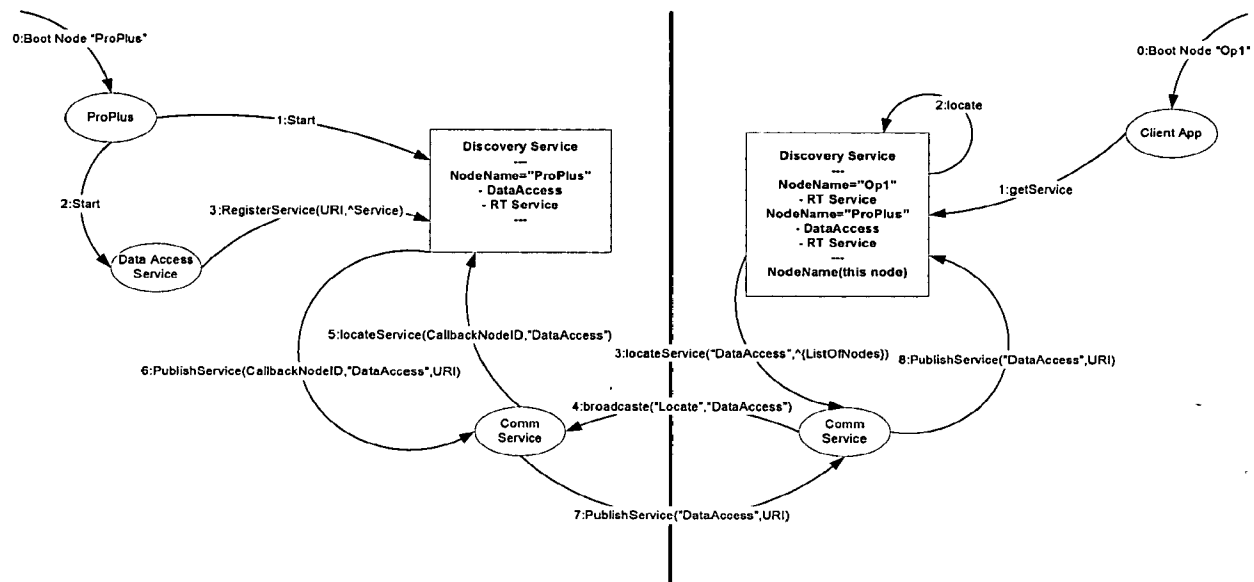


Figure 48. Discovery Service - Locate Service

Zones

Zones publishing service

Gateway device

6.3.2 Session Service

Session is a complex service. The part of session described here loads into each client and tracks the privileges for that application/user combination. The actual session object is held and controlled by the runtime. This separation allows for running clients [at least in Debug mode] without the runtime being available.

The Session Service model is shown below.

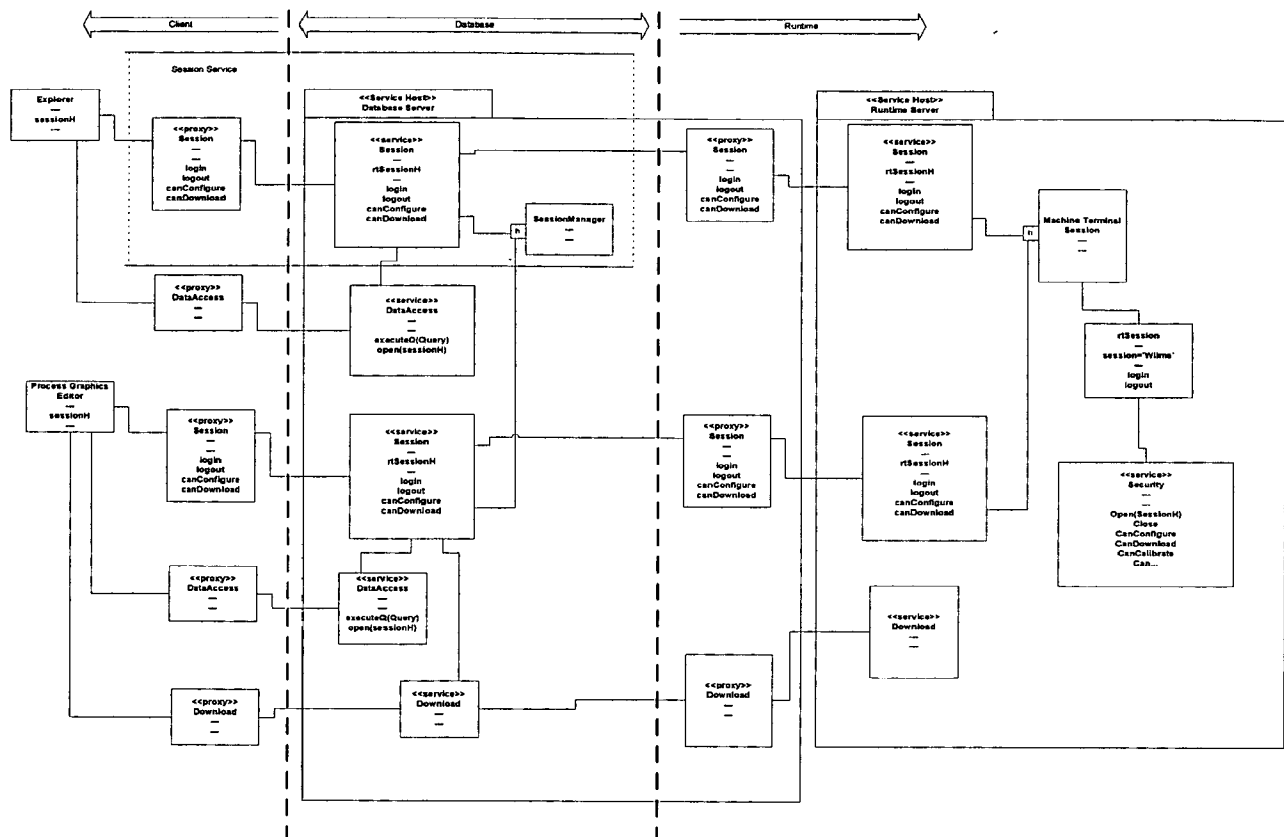


Figure 49. Session Service

These services are described below. Scenarios are then presented illustrating how the services are used.

6.3.2.1 Service Descriptions

6.3.2.1.1 Session

The session service provides an interface for logging in/out of DeltaV.

6.3.2.1.2 Session Manager

Session Manager provides a single session object for all services referenced by the same application.

6.3.2.2 Scenarios

6.3.2.2.1 User Logs In as 'User1'

6.3.2.2.2 User Launches Application 'DeltaV Explorer'

6.3.2.2.3 User Launches Application 'Process Graphics'

6.3.2.2.4 User Logs In as 'User2'

6.3.2.2.5 User Logs Out

6.3.3 Runtime Services

The Runtime Services provide access to the local and remote runtime information. The...

The Runtime Service model is shown below.

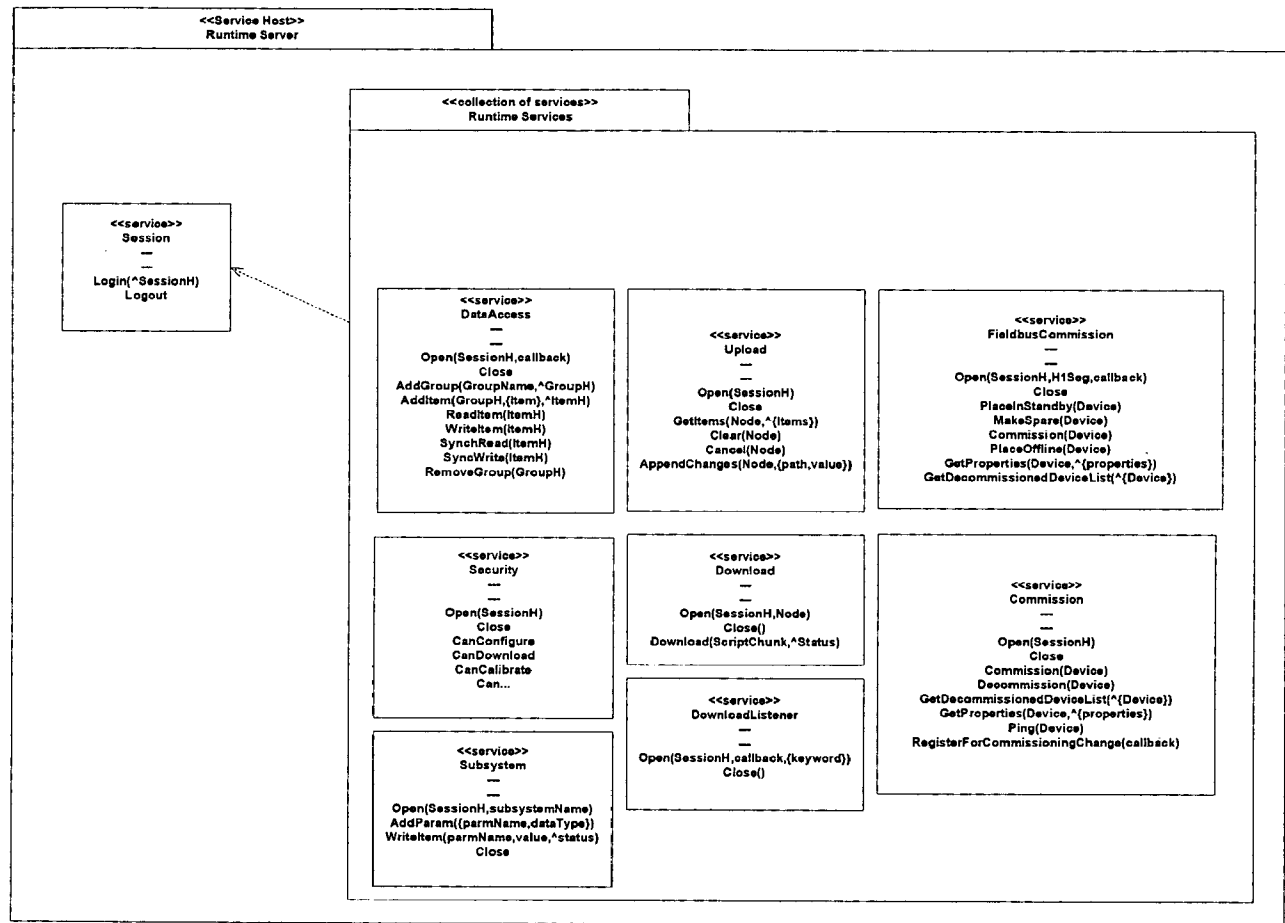


Figure 50. Runtime Services

These services are described below. Scenarios are then presented illustrating how the services are used.

6.3.3.1 Service Descriptions

6.3.3.1.1 Data Access

The runtime data access service provides an interface to allow the reading and writing of parameters in the runtime database, and to enumerate certain collections of items in the runtime database such as commissioned controller nodes.

1. Clients need to form a session associated with the current user. When the session is formed a session handle is returned (^sessionHandle).

2. The runtime supports the following functions:

- Locate (modules, DSTs, etc) – get back a server handle to the item that has been located
- Register for change notifications
- Synchronous calls
- Download / Upload
- Etc

3. Support local and remote data through a local service

4. Support multiple items in lists

6.3.3.1.2 Administration

The runtime administration service provides an interface to allow commissioning and decommissioning of devices.

6.3.3.1.3 Configuration

The runtime configuration service provides an interface to all configuration download.

6.3.3.2 Scenarios

6.3.3.2.1 Scenario 1

6.3.4 Database Services

The overall database services are shown in the drawing below:

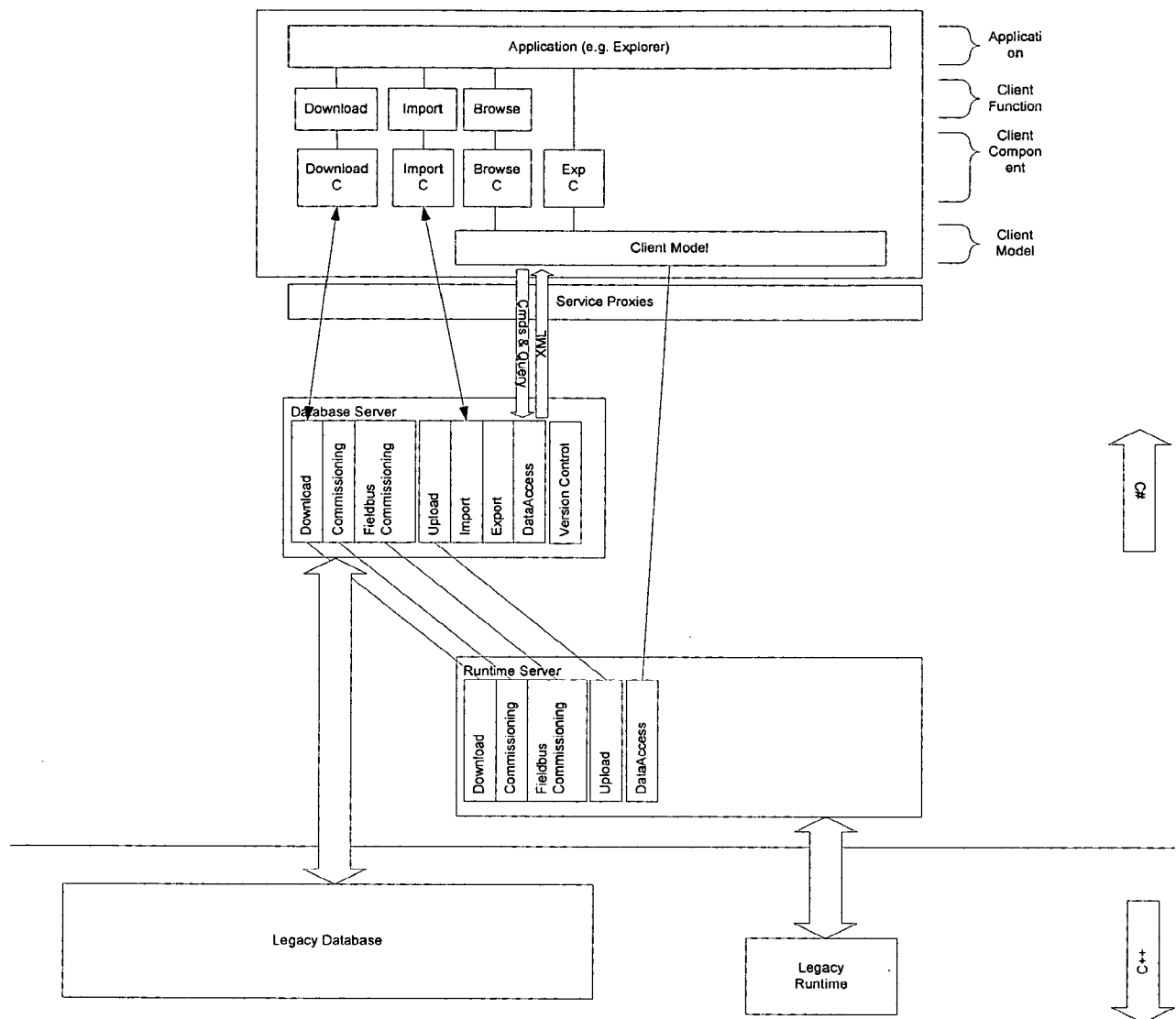


Figure 51. Configuration Architecture

There will be a database server, equivalent to today's DvDbServer. The following are the primary interfaces available on this server.

The Database Service model is shown below.

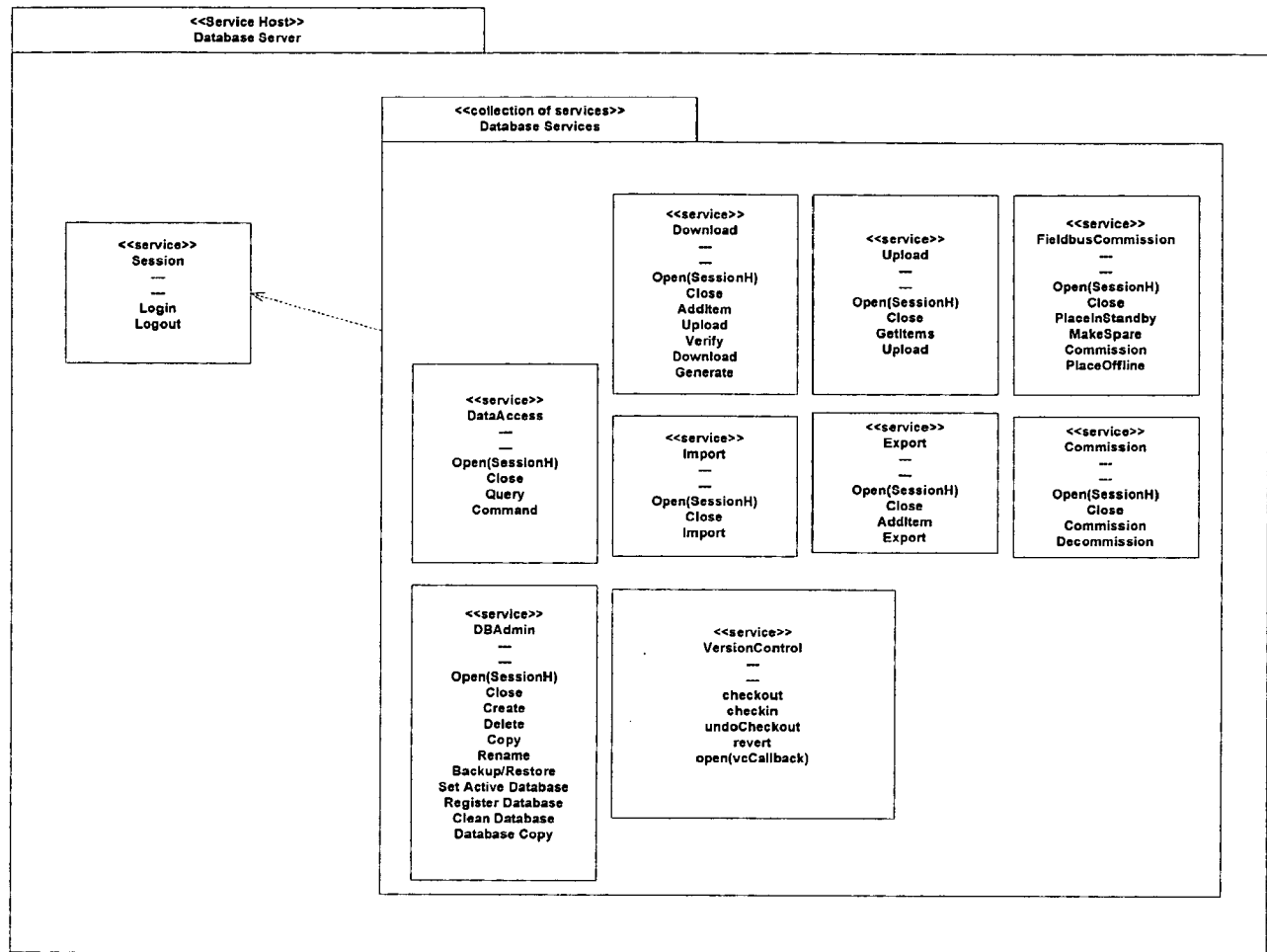


Figure 52. Database Services

These services are described below. Scenarios are then presented illustrating how the services are used.

6.3.4.1 Service Descriptions

6.3.4.1.1 Data Access

The data access interface contains functions to query and update the database.

The query function allows the database to be queried using our query syntax and the result return as an XML stream which is interpreted by the client model.

The command function allows commands issued by a client to be executed against the database in the server. A single command is made up of multiple navigation and update operations against the database and executes in a single database transaction. A command is composed in the client and sent as a string which is interpreted as a script in the server. A command will usually be parameterized so that a single script can be used for all similar operations. The server may cache and pre-compile scripts as a performance optimization. Any update command can trigger a callback onto a client interface if version control actions are required as a result of the command executed.

6.3.4.1.2 Download

Download will be implemented as a sequenceable interface in the server. The interface to download will include instructions to add items to a download list; to check for and perform uploads, to check dependencies, to verify the download and to actually send download items to the DeltaV system.

This service will make use of the Runtime Data Access service to perform uploads, the runtime Administration service in order to ensure that devices are commissioned and the Runtime Download Service to actually send the download scripts.

6.3.4.1.3 Export

Export will be implemented as a sequenceable interface in the server. The interface to export will include instructions to add items to an export list and then to perform the actual export. There will be a callback interface on the client to write the export file for the server and to report the progress of the export.

6.3.4.1.4 Import

Import will be implemented as a sequenceable interface in the server. The interface to import will include instructions to specify the import file and import options and then to perform the actual import. There will be a callback interface on the client to read the import file for the server and to report the progress of the import.

6.3.4.1.5 Version Control Administration

A version control interface interface provides instructions to administer version control.

6.3.4.1.6 Database Administration

An administration interface on the server will give access to database administration functions; such as creating, deleting, backing up and restoring databases.

6.3.4.2 Scenarios

6.3.4.2.1 Scenario 1

6.3.5 Version Control Service

The version control service provides an interface to allow storing and retrieving version information information. The Version Control Service model is shown below.

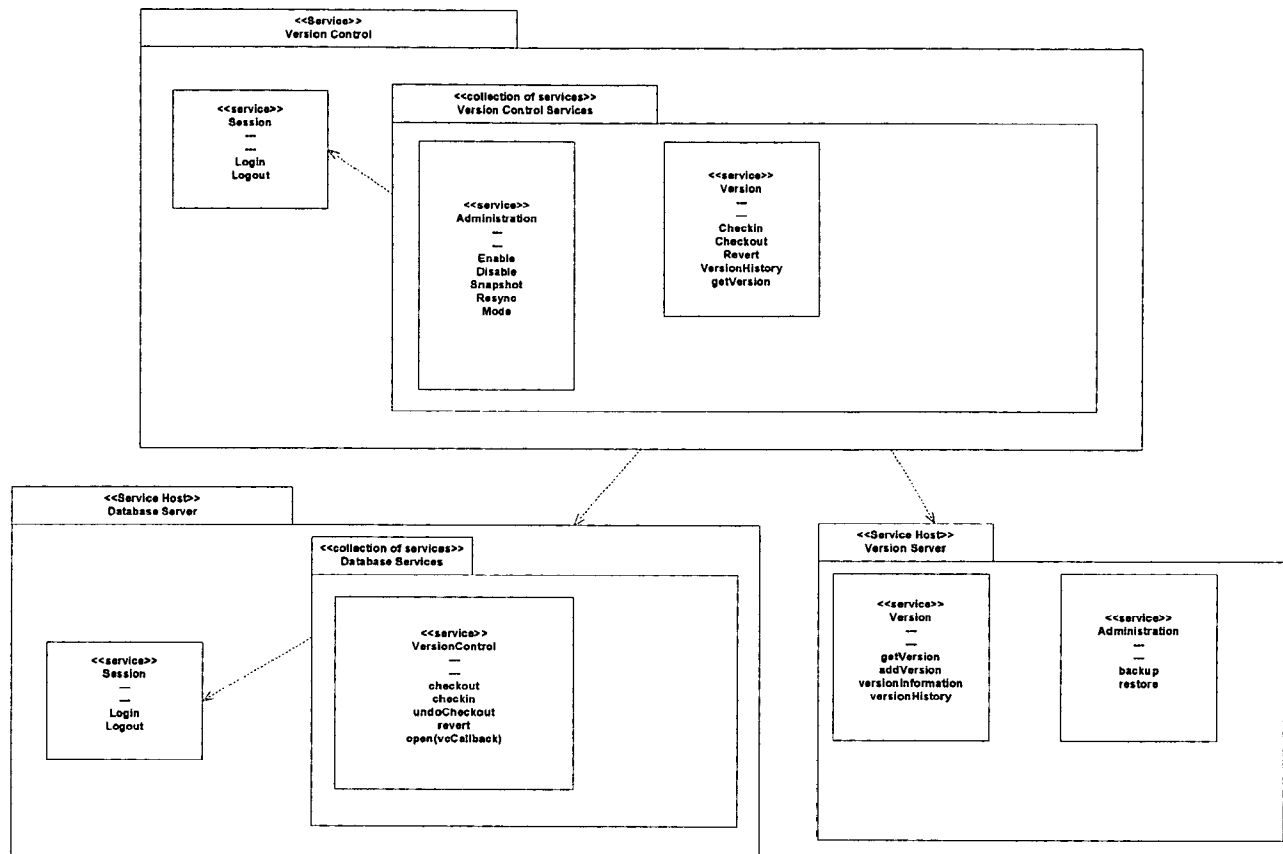


Figure 53. Version Control Server

These services are described below. Scenarios are then presented illustrating how the services are used.

6.3.5.1 Service Descriptions

6.3.5.1.1 Session Service

6.3.5.1.2 VersionControlServices.Administration

6.3.5.1.3 VersionControlServices.Version

6.3.5.1.4 DatabaseServer.VersionControl

6.3.5.1.5 VersionServer.Version

6.3.5.1.6 VersionServer.Administration

6.3.5.2 Scenarios

6.3.5.2.1 Scenario 1

6.3.6 Historian Service

The history access service provides an interface to allow querying for historical information

The Historian Service model is shown below.

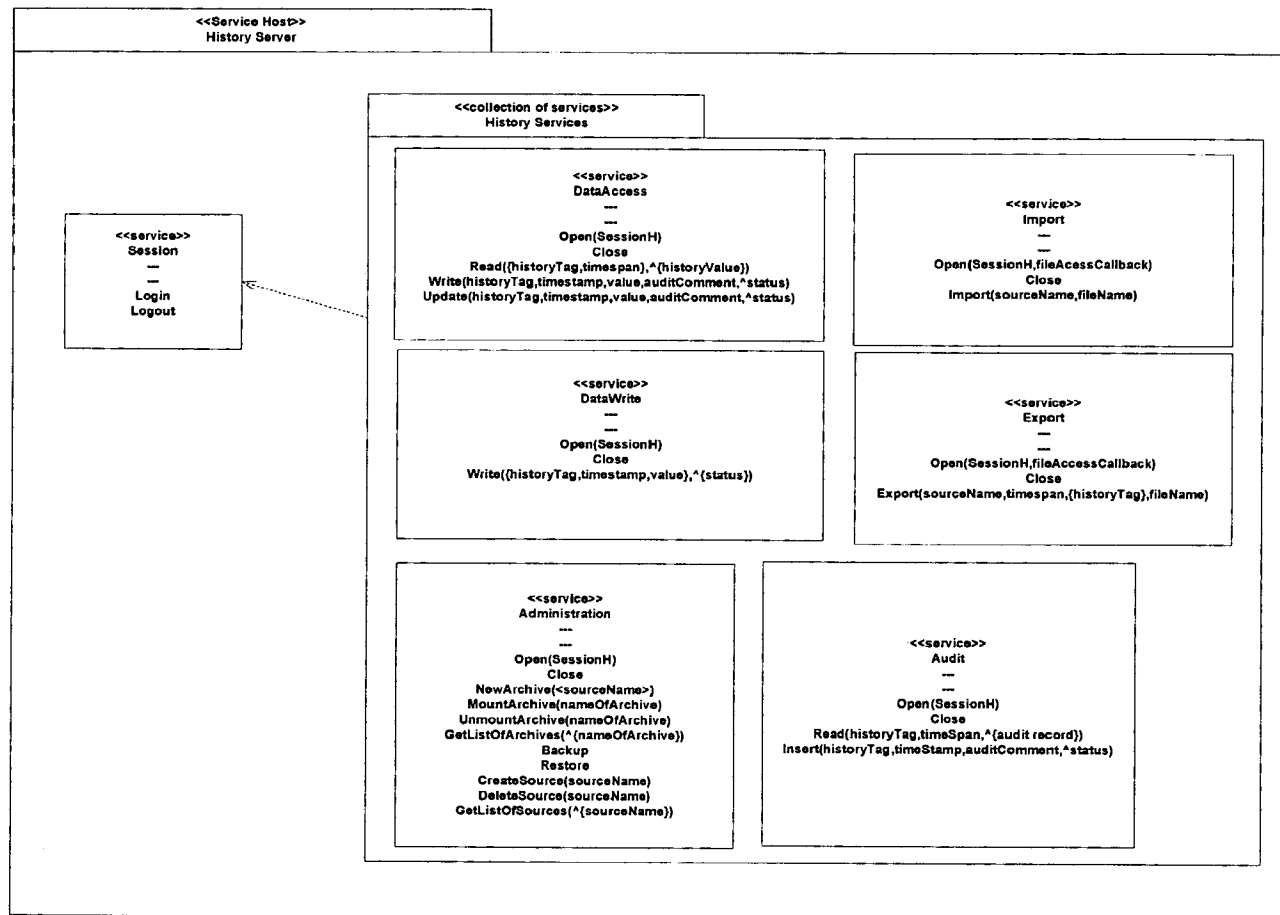


Figure 54. History Server

These services are described below. Scenarios are then presented illustrating how the services are used.

Service Descriptions

6.3.6.1.1 Data Access

6.3.6.1.2 Data Write

6.3.6.1.3 Administration

The history administration service provides an interface for administering history collection, including creating, deleting and moving archives of historical data.

6.3.6.1.4 Audit

6.3.6.1.5 Import

6.3.6.1.6 Export

6.3.6.1.7 Configuration

The history configuration service provides an interface to allow history collection characteristics to be specified.

6.3.6.2 Scenarios

6.3.6.2.1 Scenario 1

6.3.7 Historian Scanner Service

The historian scanner service does not provide any services. It requires services from Session, Runtime and Historian.

The Historian Scanner Service model is shown below.

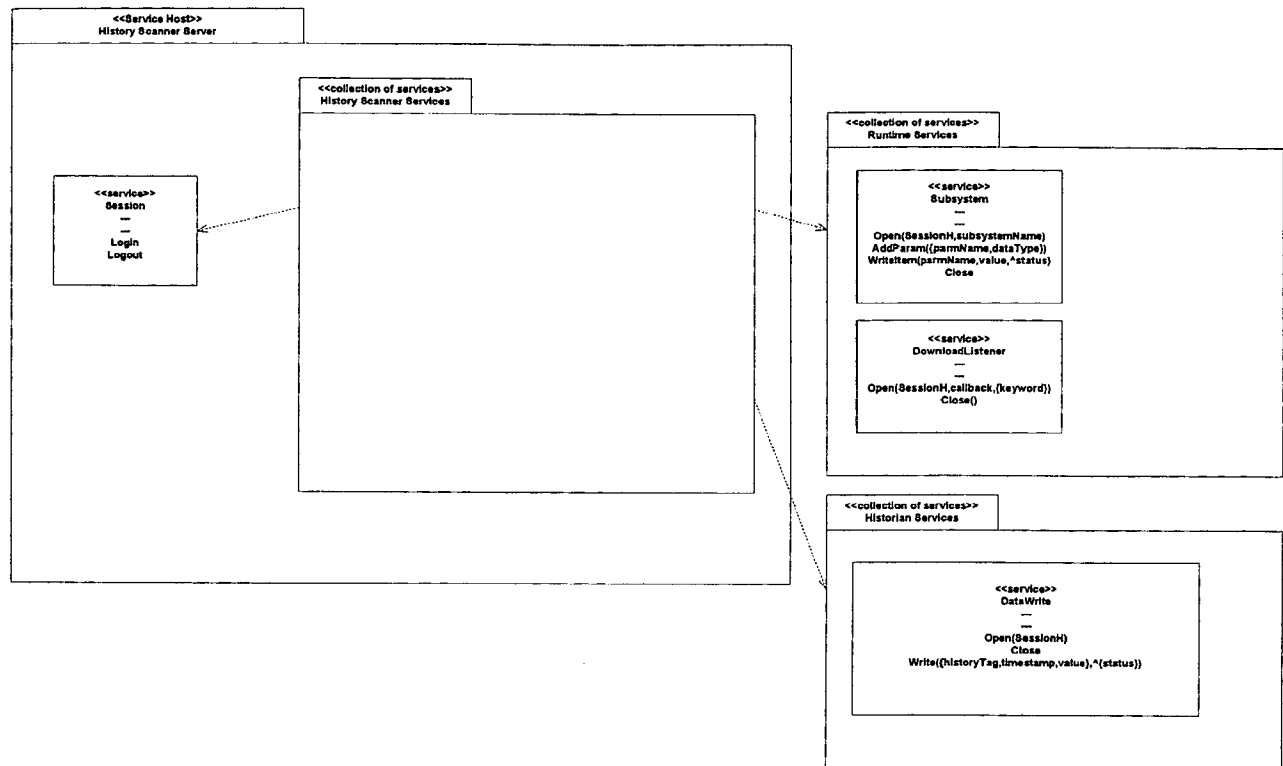


Figure 55. History Scanner Service

These services are described below. Scenarios are then presented illustrating how the services are used.

6.3.7.1 Service Descriptions

6.3.7.2 Scenarios

6.3.7.2.1 Scenario 1

6.3.8 Alarms and Events Services

The alarms and events data access service provides an interface to allow the reading alarm information from the alarms and events server.

The Alarms and Events Service model is shown below.

These services are described below. Scenarios are then presented illustrating how the services are used.

6.3.8.1 Service Descriptions

6.3.8.1.1 Data Access

6.3.8.1.2 Administration

The runtime administration service provides an interface to allow commissioning and decommissioning of devices.

6.3.8.1.3 Configuration

The runtime configuration service provides an interface to all configuration download.

6.3.8.2 Scenarios

6.3.8.2.1 Scenario 1

6.3.9 OPC UA Data Services

The OPC UA data access service provides an interface to allow the reading and writing of parameters in the runtime database.

The OPC UA Service model is shown below.

These services are described below. Scenarios are then presented illustrating how the services are used.

6.3.9.1 Service Descriptions

6.3.9.1.1 Data Access

6.3.9.1.2 Administration

The runtime administration service provides an interface to allow setting up the OPC interfaces.

6.3.9.1.3 Configuration

The configuration service provides an interface to configure OPC Servers.

6.3.9.2 Scenarios

6.3.9.2.1 Scenario 1

6.3.10 XML Files Service

The XML file service provides an interface to read from XML formatted files.

The XML Files Service model is shown below.

These services are described below. Scenarios are then presented illustrating how the services are used.

6.3.10.1 Service Descriptions

6.3.10.1.1 Data Access

6.3.10.2 Scenarios

6.3.10.2.1 Scenario 1

6.3.11 HTTP Link Services

The HTTP service provides an interface for reading and navigating HTTP Links.

The HTTP Service model is shown below.

These services are described below. Scenarios are then presented illustrating how the services are used.

6.3.11.1 Service Descriptions

6.3.11.1.1 Data Access

6.3.11.2 Scenarios

6.3.11.2.1 Scenario 1

6.3.12 Server versioning

There is a separation between server interface versions, server schema versions and server build versions.

- If a server is updated as a bug fix, then the server build version is changed.
- If the server is updated to add new database schema components (new items to be configured), but the interface to the database service is not changed, then the server schema and the build versions are changed.
- If new functions are added to existing server interfaces, or new interfaces are added to a server then the server version is changed also.

Generally a server build version change should require no change to any client applications.

A server schema change, providing it is only adding to the existing schema should not require that existing client applications be updated. Schema changes which modify existing schemas should be avoided if at all possible. If schema changes occur they will affect applications using that portion of the schema.

A server interface change, providing that it is adding to existing functions on an interface, or adding new interfaces to the server should not require that existing client applications be updated. Interface changes which modify an existing interface should be avoided if at all possible. If interface changes do occur they affect applications which use the affected functions.

This behavior means that for the vast majority of server changes, existing applications should continue to work with new versions of a server.

6.3.13 Client interface to server

For each server interface there is built a reusable client interface assembly which encapsulates client access to that server. Client applications will use these client assemblies rather than communicating directly with the service interfaces of the server. Client interface assemblies will be versioned, corresponding to versions of the server interface. A client machine can have multiple versions of a client interface assembly active simultaneously; each application will be bound to versions of the client interface assembly that it expects.

6.4 A service example

The following discussion briefly describes services. Examples are used to illustrate concepts described in this document.

6.4.1 Framework

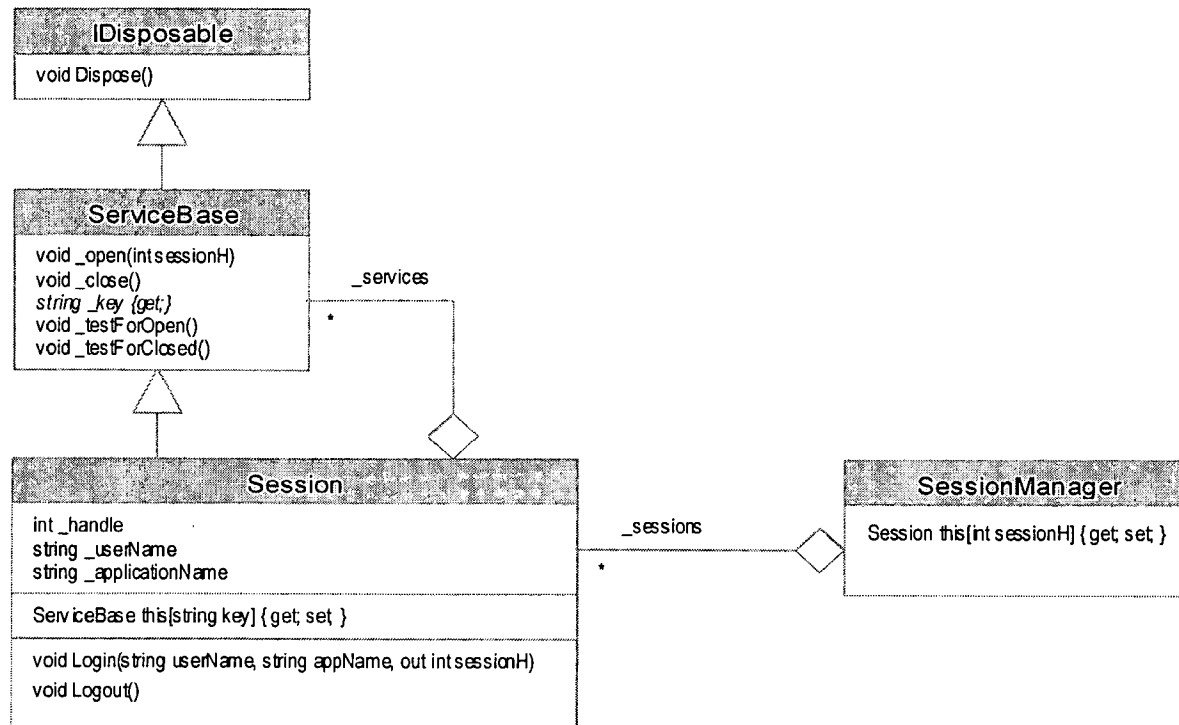


Figure 56. Service Architecture Framework

6.4.1.1 ServiceBase

All Starburn services inherit from **ServiceBase** and with the exception of the **Session** service all services must have an **Open** and a **Close** method.

The minimal implementation of **Open** is shown below

```

[ServiceMethod]
public void Open(int sessionH)
{
    _testForClosed();
    _open(sessionH);
}
  
```

The minimal implementation of **Close** is shown below

```

[ServiceMethod]
public void Close()
{
    _testForOpen();
    _close();
}
  
```

Every service must implement the `_key` property to return a string uniquely identifying the service with the server.

Every server other than `Session` must call `_testForOpen()` at the beginning of a server method or message.

Service classes are decorated with a custom attribute indicating the kind of service they provide, the name and namespace, for example

```
[DialogPortType(Name = "DataAccess", Namespace = "http://tempuri.org/DeltaV/Database/DataAccess")]  
public class DataAccess : ServiceBase
```

This will produce a service description `IDataAccess` in the namespace `DeltaV.Database.DataAccess`

6.4.1.2 Session

Every Starburn service server will include a `Session` service. This is a special service used to coordinate the other services within the server and validate security credentials.

A client application connects to the server at a specific URI and asks for the `Session` service interface. The `Session` service has two service methods `Login` and `Logout`.

`Login` will perform the necessary security validation and establish (or connect to) a runtime session for the caller. A session handle is returned.

When the client opens another service interface within the server it must pass in the session handle. Internally the new service is added to the session. At logout any services used by the client that have not been explicitly closed are closed.

6.4.1.3 SessionManager

The `Session Manager` keeps track of active sessions within a host.

6.4.1.4 Proxy

Each service must provide a proxy to be used by client applications that interact with the service. The proxy is generated automatically from the service definition using the `wsdlgen.exe` tool.

A service is built into a .net assembly. The service assembly project has a post build step that generates the service description `.wsdl` and `.xsd` files into the service source code directory.

`BuildServiceDefinition.bat` below demonstrates how the service definition for the `Session` service is generated.

```
@echo off  
d:  
cd \DeltaV\code\Starburn\Session  
"%SDKTOOLPATH%\wsdlgen.exe" /nologo D:\deltav\code\bin\DeltaV.Session.dll  
if errorlevel 1 goto ReportError  
goto End  
:ReportError  
echo Project error: A tool returned an error code from the build event  
exit 1  
:End
```

This generates the following files

- tempuri_org.DeltaV.Session.wsdl
- tempuri_org.DeltaV.Session.xsd
- schemas_microsoft_com.serialization.2003.02.DefaultDocumentElement.xsd

A Proxy project is built in the directory below the service. The project contains a prebuild step that generates Proxy.cs from the service description in the directory above. This is compiled into the service proxy assembly.

```
"%SDKTOOLPATH%\wsdlgen.exe" /nologo /o:$(ProjectDir)Proxy.cs $(ProjectDir)..\tempuri_org.DeltaV.Session.wsdl
$(ProjectDir)..\schemas_microsoft_com.serialization.2003.02.DefaultDocumentElement.xsd
$(ProjectDir)..\tempuri_org.DeltaV.Session.xsd
```

6.4.2 Database Services

Database services are hosted by DvDatabaseServer.exe, they include DataAccess and Download as shown in **Error! Reference source not found.**

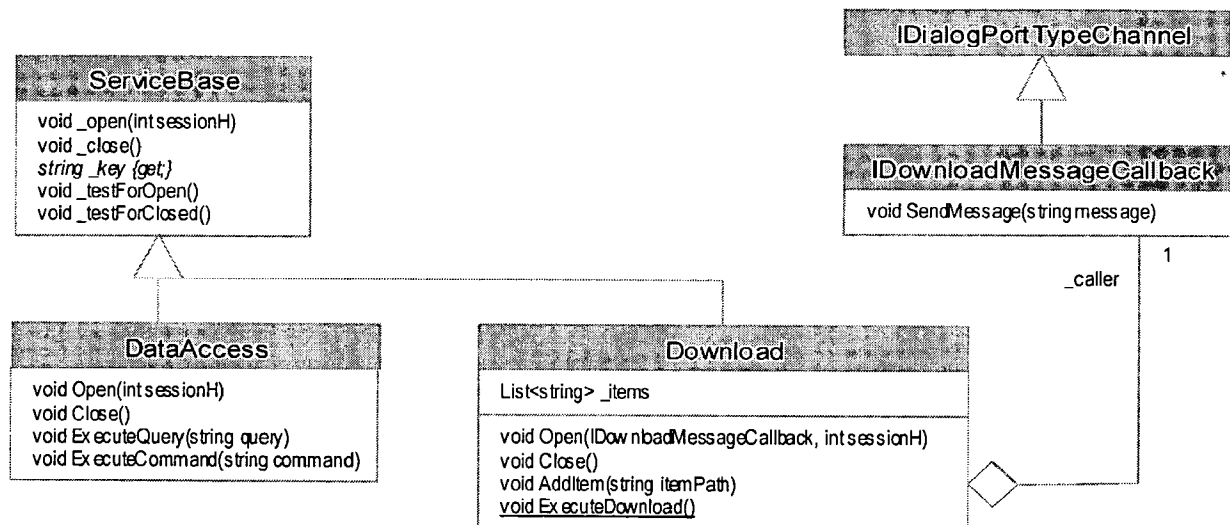


Figure 57. Database Services

6.4.2.1 DataAccess

The DataAccess service provides a general query and command interface into the database.

6.4.2.2 Download

The Download service manages the download sequence. The service methods in **Error! Reference source not found.** are illustrative of those required for download. Clients of this service must first call AddItem to add items to be downloaded before calling ExecuteDownload.

ExecuteDownload is a service message instead of a service method. This means it executes asynchronously. As it runs this method uses the callback interface (IDownloadMessageCallback) to provide the client with feedback. The client implements this interface and it is passed into the download service on Open.

6.4.3 Servers

Services are deployed in servers, for example the database server (DvDatabaseServer.exe) the runtime server (DvRuntimeServer.exe) and the historian server (DvHistorianServer.exe).

Each server contains an Indigo port identified by a unique URI. When the server is started it registers its URI with the local DeltaV discovery service. As client applications come up they ask the discovery service for the URI of whichever servers they need. They can then establish an Indigo channel connecting them to the server port. Once connected the client application can request a specific service interface from the server.

6.4.4 Service Clients

Error! Reference source not found. shows a typical client application connecting to the database and runtime servers. The client references the proxy assemblies for the services it intends to use from these servers.

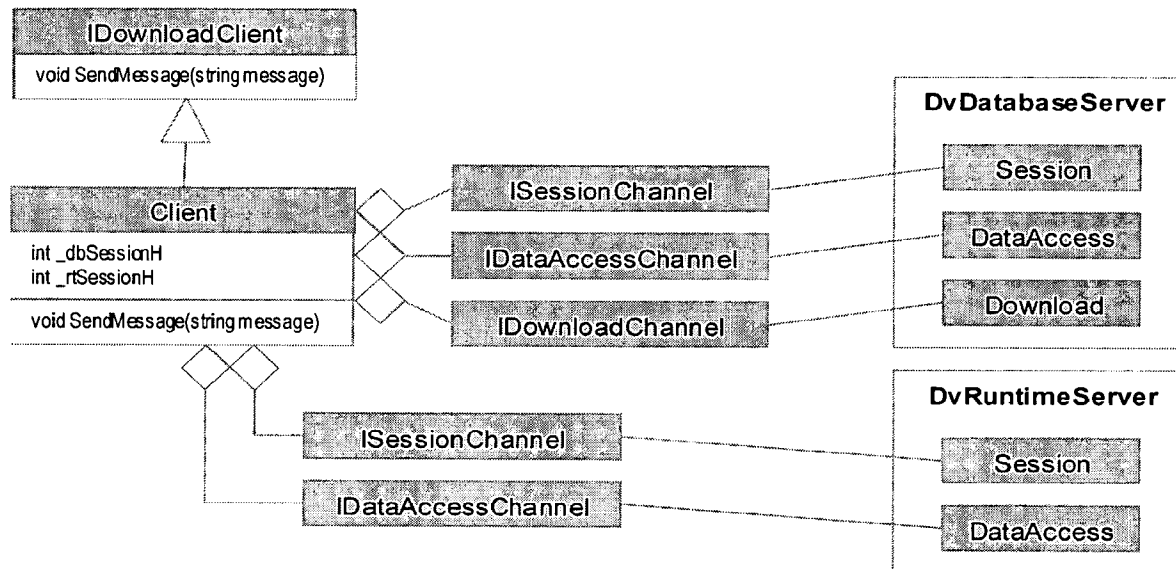


Figure 58. Service Client

The first service the client uses will be the Session service, on which it will call Login. The client must do this for each server it intends to use. In **Error! Reference source not found.** the client holds session handles for the database and runtime servers.

After calling Login the client can open other services and use them. Some services will require a callback interface on Open, for example the Download service. For download the client must implement the callback interface as defined in the service proxy and pass itself as the target of the callback on connecting to the service.

The pseudo code below indicates the sequence of events.

```
class DatabaseTest : IDownloadClient
{
    [STAThread]
    static void Main(string[] args)
    {
        DatabaseTest test = new DatabaseTest();
    }
}
```

```

// Open the indigo service environment
ServiceEnvironment se = ServiceEnvironment.Load();
ServiceManager manager = se[typeof(ServiceManager)] as ServiceManager;
se.Open();

// Create a port for callbacks
Uri portUri = new Uri("soap.tcp://localhost/Database/");
Port port = new Port(portUri);

// Get the URI of the database server
Uri serviceUri = new Uri("soap.tcp://localhost:46001/Database/");

// Get the session service from the database server and login
ISessionChannel session;
session= (ISessionChannel)manager.CreateChannel(typeof(ISessionChannel), serviceUri, test);
int sessionH;
session.Login("Bill", "DatabaseTest", out sessionH);

// Get the data access service from the database server and open it and execute a query
IDataAccessChannel dataAccess;
dataAccess = (IDataAccessChannel)manager.CreateChannel(typeof(IDataAccessChannel), serviceUri, test);
dataAccess.Open(sessionH);
dataAccess.ExecuteQuery("Site.Modules");

// Close the open services and logout, or just logout
session.Logout();

}

// IDownloadClient
public void SendMessage(IDownloadChannel sender, string message)
{
    Console.WriteLine(message);
}
}

```

6.5 Proof of Concept

In order to prove the goals for isolation have been met the following tests will be run

- Layer process graphics onto Ovation, this requires Ovation to provide an implementation of the runtime service and a browser plug-in.
- Run process graphics on Provox
- Run the browser on OS file system

6.6 Topologies

The following Topologies will be supported

- 1- Single stand-alone laptops
- 2- Small Systems

- 3- Large Systems
- 4- Zones
- 5- Remote Clients
- 6- Browsers

6.7 Key Technologies

The following technologies will be leveraged:

- 1- Existing DeltaV System
 - a. Runtime
 - b. Database
 - c. Control
 - d. Security and Session
 - e. Alarms and Events
- 2- Longhorn
 - a. Indigo
 - b. Avalon
- 3- Compiler
 - a. Whidbey
- 4- Simulation
 - a. High Fidelity Models
 - b. FIR Technology
- 5- Expert
 - a. CLIPS

6.8 Versioning

Services are registered on each node. The services register as the node boots and servers are started. Services fall into three categories:

- 1- Services where there is 1 service in the system.
- 2- Services that exist on every node.
- 3- Services that exist on nodes if they are configured and licensed.

Here are things that we can do to services:

- 1- Extend the interface
- 2- Version the interface
- 3- Rebuild/recompile the implementation for a new compiler and/or OS

- 4- Recompile the interface with a new compiler (new OS)

Services need to be versioned. We version services so that we can maintain upward and downward compatibility.

Clients see the following information:

- 1- preferences
- 2- licenses
- 3- version information

6.9 Deployment

- 1- Clients can support multiple versions of services.
- 2- Services can support multiple versions to clients.
- 3- A version number will be included in the XML exchange when the service is located.
- 4-

6.10 Session

6.11 Security

6.12 Globalization and Localization

6.13 Usability Guidelines

The configuration Framework and subsequent applications developed should have a look / feel modeled after Visual Studio and Windows Explorer

7 Project Architecture

7.1 Overview

The topics in this document have, up to this point, discussed the newer functionality of Starburn and the frameworks, patterns, and techniques that will be used to both incorporate that functionality and build a scalable high performance set of applications and services. This chapter moves on to address the details required to build this project with a large team spread across several development sites. It discusses in detail structure of the development environment, software configuration management and versioning, team organization, and deployment. The chapter is organized into the following sections:

- 1- **Concepts** – the main concepts needed to understand the Project Architecture.
- 2- **Software Configuration Management and Version Control** – this section describes concepts, development process, and integrated software configuration management from the point of view of our services approach.
- 3- **Dependency Management** – this section introduces a dependency model, which includes a categorization of dependencies between components, and a model for managing these dependencies.
- 4- **Development Environment and Deployment** – completes the description of the development environment to include details such as how we build, transfer builds, and deploy software.
- 5- **Testing** – discusses the approach for testing services, applications, and versioning.
- 6- **Project Manager Model** – discusses the organizational principles of architecture-centric development, various project management views including the role of the architecture team, and the organization of teams around the services and applications that have been defined.

7.2 Concepts

The discussions presented so far in this document describe a Technical Architecture and an Application Architecture organized along the boundaries of services and functional groupings⁴². Functional groupings can be as small as a something like a diagram control or drag and drop or as large as entire application such as Process Graphics. As presented earlier in this document, we have learned how to develop major new functionality concurrently and iteratively – we know how to build and release using object-oriented technology. We have also become very good at testing and releasing very robust, high quality software. What has eluded us, however, is the ability to version software artifacts of anything smaller than the whole system⁴³. The architectures described here address this and several other issues. From a Project Architecture point of view the principles to support this architecture are:

- 1- Services and functional groupings (i.e. software artifacts) are the basis of the Starburn approach. Project plans, builds, team structures, organizational boundaries, and task assignments recognize these service and functional grouping boundaries.
- 2- Each functional software artifact and its specifications is owned by some group. Analysis, design, coding, and deliverables recognize these software artifact boundaries.
- 3- The ownership concept is supported and strengthened by strict accessibility constraints throughout the development cycle. If an application needs to use an artifact that artifact must be made available through an explicit interface.
- 4- Autonomous development requires the reduction of surface area across the lifecycle between artifacts. Dependencies between artifacts must be carefully managed.

⁴² Services and functional groupings are sometimes referred to as ‘software artifacts’ in this document.

⁴³ This isn’t completely true – we have always been able to release a small set of system independent features.

- 5- Deliverables are organized along the boundaries of services, functional groupings, and applications.
- 6- Versioning is at the software artifact level.⁴⁴
- 7- It must be possible to develop each service, functional grouping, and application with minimal waits from other parts of the system. This implies that it must be possible to add new functionality in parallel (note – we have done an outstanding job of this in DeltaV)⁴⁵.
- 8- It must be possible to build and release new versions of services, functional groupings, and applications without completely re-testing the entire system. In cases where the changes are pervasive, it must be possible to regression test all interfaces and functional groupings.

7.3 Software Configuration Management and Version Control

Software Configuration Management and Version Control are critical in being able to build and deploy software into the field. In order to version software artifacts some knowledge of the dependencies between artifacts must be known. Two classifications of dependency must be known

- 1- detailed dependency list – all software artifacts that a particular software artifact depends on
- 2- versioned dependency list – version 'x-y-z' of software artifact 'aaa' depends on these versions of these software artifacts

There is yet another level of complexity – DeltaV has both distributed components as well as layered components. As the development team knows, the existing core DeltaV system is strongly reuse based – there are no plans [and few justifiable reasons] to change this. This does lead, however, to a hierarchy of dependencies that must be understood, documented, and verifiable. For each version of software artifact that is released to the field it must be possible to document the set of dependencies as well as the known set of versions for all of those dependencies that the software artifact were tested and verified against. While the software configuration management software must be able to manage the versions of all of the core DeltaV System as well as all of the services, functional groupings, and applications built on top of the core, another mechanism is required to track dependencies of released software – this is referred to as 'Dependency Management'. Dependency Management is covered in a later section in this chapter⁴⁶.

So what about software configuration management? As we have always done SCM will be used to store and track source code. What is more complex is how the build procedures will change to accommodate both distributed development and versioning. The development environment and deployment is covered later in this document.

7.3.1 Software Configuration Management

Software configuration management is well known to the DeltaV team⁴⁷. We have addressed large scale development and releases through a combination of labeling, branching/pinning, and development procedures. The use of SourceSafe has been integrated with the IDE for many releases. Building versions of DeltaV has amounted to pulling a complete copy of source code from a specific branch each night, running the build procedures, localization routines, generating install images, and transferring the whole set to a file server. We do not propose that this procedure change for the core DeltaV infrastructure.

⁴⁴ The first question we ask customers who call up with a problem is "What version is your system?" With the additional flexibility of versioning at the artifact level and large numbers of artifacts being present in the system, the answer will not be as simple as it used to be. We will need to provide an application enabling both customers and our support staff to easily view and understand the version information associated with all their artifacts.

⁴⁵ Faking and simulating components and interfaces that are not yet available is a technique that has worked well to enable parallel development.

⁴⁶ A database will need to be created to support Dependency Management

⁴⁷ At this point we are not actively pursuing a replacement for SourceSafe. I would like to see what features are available as part of the next version of Whidbey.

We do, however, plan to make significant enhancements to both SCM as well as the build procedures to support multi-site development and versioning. For starters, different SS databases will be used to manage services, applications, and other software artifacts that can be independently versioned. To make source code integration easier to manage separate databases will also be used for Austin, Leicester⁴⁸, Manila, and Pittsburg.

To address versioning the build system will be able to build specific versions of services, software artifacts, and applications. This will be achieved using a combination of a database for tracking dependencies and procedures for pulling the right bits out of source. To build install images it will be necessary to combine these version builds with a "golden" disk repository to fully assemble the complete set of deliverables. This ad-hoc combination of SCM, dependency database, and golden disk repository is required to fully support the deployment requirements set forth in this document.

At this point at least a couple of questions should be in everyone's minds:

- 1- Where is the "golden" copy of each software artifact stored? As far as source code developed by the developer is concerned, the software is stored in SCM. Code generated by the tools, e.g. Schemas and Proxy's, are stored on the file system. The golden copies of all deliverables released to the field are stored in the file system.
- 2- Which level of version granularity will be known to the tools? Versions of files developed by developers are obviously always stored in the SCM. Versions of assemblies to be included in specific software deployments must be managed by the dependency management system.

A point that deserves repeating is this – the versionable items in the DeltaV System are services, applications, and supporting software artifacts that can be compiled into assemblies or exe's.

Based on this discussion the SCM strategy will be:

- 1- The SCM system can be used for check-in and check-out. Within the SCM system software services, software artifacts, and applications are grouped by their respective function.
- 2- Each version of each individual software service, software artifact, and application as well as the dependencies between them is managed outside of the SCM system. A dependency management system is used to track version dependencies. A file system is used to store all of the versions. Strict guidelines are used to govern how a physical file system directory structure is to be used store all of the released versions.
- 3- The SCM system is organized around the DeltaV Core Infrastructure, services, software artifacts, and applications.
- 4- Each software development site has its own SCM System⁴⁹.
- 5- The Austin build procedures will be able to perform a complete build of core infrastructure, services, software artifacts, and applications. A complete build includes all the steps necessary to build install images.
- 6- The Austin build procedures will be able to perform version builds. Version builds include the steps necessary to build install images for minor releases, service pack releases, and hot fixes.

7.3.2 Versioning

New applications using updated library components run concurrently with existing applications using older versions of library components. This is made possible by using the versioning capabilities of .NET modules.

⁴⁸ The exception here is that database code will continue to be integrated into the core DeltaV infrastructure

⁴⁹ Consistent software and procedures across sites but, different content.

This is especially critical in deploying application and core service versions independently. When a new version of a service is installed it comes with a new version of the proxy to be used by the applications. Unless the newer version of an application is installed it will continue to use the older version of the service proxy. Thus both versions of the service proxy may need to reside on the client machine at the same time.

7.3.3 SCM System Layout

The following sections describe the layout of the configuration management system.

7.3.3.1 Core Infrastructure

```

$\DeltaV
  \Code
    \bin (this arrangement is still under construction)
      \DvRuntimeServer
        .exe, .dll, .manifest, .deploy
  \v1.1
    .exe, .dll, .deploy
      \
        \Starburn (interop)
          \DvDatabaseServer
            \ConfigXM
  \Test
    \Docs
  \DvRuntimeServer
    \Security
    \Session
    \DataAccess
    \Test
    \Docs

```

7.3.3.2 Austin

```

$\Starburn
  \Core
    \Mainline
      \Session
        \Proxy
        \Test
      \DvDatabaseServer
        \DataAccess
          \Proxy
        \Download
          \Proxy
        \Test
        \Docs
      \DvRuntimeServer
        \DataAccess
          \Proxy
        \Session
          \Proxy

```

```

\Test
\Docs
\DvCorLib
\Doc
\BrowserFramework
\Mainline
\1.1

\ClientModel
\Mainline
\Model
\Schema
\Browser
\1.1
\Model
\Schema
\Browser
\DvControlStudio
\Mainline
\1.1

```

7.3.3.3 Manila

```

$\Starburn
\DvProcessGraphicEditor
\Mainline
\Browser
\1.1
\Browser
\DvProcessModuleEditor
\Mainline
\1.1

```

7.4 Dependency Management

Simply stated, Dependency Management is the set of concepts and tools that allow proper management of dependencies in a service based system. Dependency management includes for each project:

- 1- The definition of a dependency model which includes the identification of possible categories of dependencies between services, software artifacts, and applications. The model also includes how the project intends to handle those dependencies.
- 2- The development of appropriate tools to manage these dependencies.

7.4.1 Dependency Model

The dependency model is an explicit description of the dependency categories between services and artifacts and how a project will deal with these dependencies. An explicit description of what is visible outside an interface or software artifact is defined. Functionality is made visible by explicitly defining it as such. The approach we are using to version software uses an explicit interface model. As an example:

- 1- A software component B “exposes” software feature “x” by designating it as available for use by another software component through an interface.
- 2- A software component A uses software feature “x” from B by referencing A in its project.

- 3- Software component A then uses software feature “x”.

The two main features in this approach are to support the reduction of surface area between software components and to support the management of unavoidable dependencies. The features are enforced by the notion of strict enforcement of ownership of software artifacts. This in turn leads to better support of change management which ultimately enhances our ability to develop and release independently.

The dependencies in a project can be summarized as follows:

- 1- **Use an Interface** : This is a runtime method invocation dependency – in order to invoke a method on a service at runtime a software artifact, service, or application must know the proxy, data types, and error definitions of the thing being invoked.
- 2- **Extend an Interface** : This is a development time interface dependency – in order to use a functional grouping in the system a component must implement an interface defined by another component. For example, in order to support Browse functionality Process Graphics must implement the Browse Interface.
- 3- **Reuse code** : This is an internal design dependency – a number of software items may be re-used in order to implement some set of functionality. In these cases the decision is being made to reuse code instead of implementing the same functionality over and over.

7.4.2 Development Environment

A given software artifact can be in several versions of DeltaV at the same time. To support this multiple versions of software will be available on the same system at the same time. How this handled for developers is different than how it done for released systems. For developers the ‘...\code\bin’ directory will be partitioned to support versions. For release deployment the GAC will be used.

The development environment is set up to allow developers to develop, debug, and unit test those parts of the system they are responsible for. By pulling enough source code the developers can also debug into and at times troubleshoot those parts of the system that they have dependencies on. A complexity that is introduced in this development is how to unit test backward versions. To do so developers also need multiple versions of services and other software artifacts where they have version dependencies.

As was discussed earlier, from a SCM point of view, as a general rule a change in a dependent component must be explicit. For example, if a software grouping or application is changed the dependency list must be updated to enforce that the new version of the service will be used. When the application is deployed the services it is dependent on must also be deployed. For developers this means that the developers need to have all of the dependent versions of assemblies duplicated in the different build directories. So how do developers deal with this?

The developer environment will be organized as a directory tree structure. As is currently done with DeltaV, the SCM system layout mirrors the file structure – this makes it easy for developers to locate projects, checkin/out files, etc. What is different from today’s structure is the location that interfaces, assemblies, applications, and software artifacts that can be versioned compile into. The approach, simply stated, is to copy all of the referenced assemblies and supporting files for each project into versioned directories. Although this leads to enormous amount of duplication the approach assures that developers will have the correct versions to work with when they are developing, unit testing, and debugging versions of items. So where is the master copy of all of all of those versions?

In order to support developers we are proposing that the build system maintain

- 1- Nightly builds as is done today.
- 2- A nightly build structure for versioned items that includes the last 21 builds.
- 3- A release build structure as is done today.
- 4- Extensions to the release build structure that includes every released version ever released. This structure must be kept around for-ever.

Developers will be able to pull current as well as previous versions of services, artifacts, and applications for development, unit testing, and trouble-shooting field problems.

The layout of the “bin” directory is shown below.

```
\bin
\DeltaV.ClientModel
  \v1
    clientmodel.dll
  \v1.1
    clientmodel.dll
\DeltaV.BrowserFramework
  \v1
    browserframework.dll
  \v1.1
    browserframework.dll
  \v1.2
    browserframework.dll
\DeltaV.ClientModel.Browser
  \v1
    clientmodelbrowserdatasource.dll
\DeltaV.Explorer
  \v1
    exp.exe
\DeltaV.Explorer.Packages
  \v1
    modulebrowser.dll
    attributebrowser.dll
```

7.4.3 Build Environment

The build environment has to address the needs of major releases, minor releases, service packs, and hot fixes. In addition, the build environment has to be able to build and track versions. This section first discusses versions and then discusses a new tool, the Component Dependency Manager.

7.4.3.1 Versions

7.4.3.1.1 Major Releases

Major number releases consist of a complete set of core system functionality together with a complete set of services and applications. In many cases the release will include multiple versions of services for backward compatibility. Applications should represent the applications at the major number that is being released. Major releases are packaged together with a complete set of user documentation, upgrade utilities, installation tools, and so forth. Major releases are fully regression tested. These releases correspond closely with today’s DeltaV releases with one significant difference – major releases also need to be backward compatibility tested.

7.4.3.1.2 Minor Releases

Minor number releases fall into two camps:

- 1- The core infrastructure is updated to address field issues, robustness issues, and performance issues. No new functionality is introduced.

- 2- New functionality that does not require additional core infrastructure support. For example the a decision could be made to release abnormal situation prevention features that do not require communication, IO or controller support. Another example could be the addition of new Process Graphics support.

In case 1 above the versionable items in the system are not changing so no dependency lists are affected. In case 2 some combination of services, artifacts, and applications may all be affected. The various dependencies lists would all need to be updated.

7.4.3.1.3 Service Packs

Service packs are designed to collect together hot fixes and minor functional releases. The intent of service packs is to make it easy for customers and our own customer support infrastructure to install collections of updates that have been tested and validated as a complete set.

In the case of service packs collections of versions will need to be brought together so that a mixed system continues to function correctly.

7.4.3.1.4 Hot Fixes

Hot fixes are designed to address a specific issue or a set of related issues. These hot fixes are released as individual repairs into the field. When a hot fix affects the version dependencies of some component then the dependent item will also need to be included in the hot fix. The dependency management system should include this information.

7.4.3.2 Dependency Manager

The dependency model requires a tool to support it – the tool is referred to as the Dependency Manager. The Dependency Manager (DM) tool needs to be able to manage, across the development cycle, the versions of the system, the versions of the interfaces/artifacts/applications, and the dependencies between them. To manage the builds and versions the DM uses a dependency list. The dependency list appears as shown in the table below:

Component	Build Item	Version	Dependency List
DeltaV.ClientModel	ClientModel.dll	V1.0	

The DM should be able to navigate the dependency list of each item in the dependency list. The knowledge of this should be contained in a dependency database. Based on the information in the dependency list the DM should be able to navigate the entire major version hierarchy. It should be able to load the DM database and generate scripts for builds.

The DM needs to have the following capabilities:

- 1- Determine dependency consistency. No circular invocations are allowed.
- 2- Starting from dependency lists, generate the appropriate scripts for the build process. These scripts are then used to build the various versions as outlined earlier in this document.
- 3- Open a given dependency list for a given component and browse the dependencies, check for consistencies, etc.
- 4- For each Component in the DM database, provide the list of available versions.
- 5- Provide revision history on the components in the DM database.

Given that we figure out how to develop the DM database and generate build scripts, there is still one dimension to the problem. The actual build environment itself must be known. Given the direction stated in this document there will be scenarios where versioned items are originating from different versions of compilers. The strategy that will be stated here is this:

- 1- The build tools can be upgraded to the next major version when DeltaV rev's a major version number.
- 2- All minor builds off the DeltaV major version will use the major versions MS and other compilers.

There is still one more major consideration – the target system. Again the working practice here will be that major operating system revisions will be accommodated with DeltaV major version numbers.

We need a couple of things here

- 1- *To develop example scripts that show how to fetch from SS and build versions.*
- 2- *To put together a SQL database to host the DM database and support the script building as detailed in these sections.*

7.5 Testing

All software developed as part of the Starburn Program comes complete with unit testing scripts. These scripts fully test all interfaces. These tests need to also be developed so that they can be combined and used for regression testing.

7.5.1 Unit Testing

There will be a full unit testing of all interfaces. The core services will be tested as will the supporting interface layers beneath the application components. Other strategies will be developed for testing the UI itself.

Unit tests must test the interface contract and semantics. It is important to provide adequate testing of the server schema and the server interface.

There will be a test framework. All tests will be run nightly. Developers are responsibly for submitting a test for any code they write.

7.5.2 Regression Testing

There will be a full regression testing of all interfaces and all supported versions as defined in the dependency database. The core services will be regression tested as will the supporting interface layers beneath the application components. Other strategies will be developed for testing the UI itself.

Regression tests must test the interface contract and semantics. It is important to provide adequate testing of the server schema and the server interface.

7.6 Project Documentation

The project documentation is primarily contained in two places:

- 1- In SS under \$/Starburn
- 2- In Select. The organization of Select DataStores and models is show in the table below.

	DATASTORE LOCATION		
Model Stereotype	Austin (Aus)	Manila (Mnl)	Leicester (Lei)
Technical Architecture Patterns and helpers that apply across the board, throughout config or throughout runtime.	<i>SbAusTechArch *</i> CommonArchitecture ConfigArchitecture RuntimeArchitecture		
Solution A particular area of functionality deliverable to the user, such as Process Graphics or Expert Modules.	<i>SbAusSolutions</i> ConfigurationWorkSpace RuntimeWorkspace ProcessModulesRuntime	<i>SbMnlSolutions</i> ExpertModulesConfig ExpertModulesRuntime ProcessModulesConfig ProcessGraphicsConfig ProcessGraphicsRuntime	<i>SbLeiSolutions</i> ContinuousHistorian
Infrastructure The Client Model and Server layers of the system.	<i>SbAusInfrastructure</i> ClientModel ConfigurationServer		<i>SbLeiInfrastructure</i> Generic Schema ServerSchema ModelImplementation XM Interface Layer
Component A component reusable in multiple contexts, such as the Diagram Control and Structured Text Edit Control.	<i>SbAusComponents</i>	<i>SbMnlComponents</i> DiagramControl	<i>LeiComponents</i> TextEditControl
Framework. A set of classes from which Component or Solution developers can inherit.	<i>SbAusFramework *</i> ConfigurationViewFramework		<i>SbLeiFrameworks</i> GenericServerModel

Items in ***bold italics*** are DataStores. Each has a list of contained models with who is responsible for them in brackets. Datastores marked with an asterisk are transmitted between sites for external referencing.
Component Interfaces are published between sites using Select Component Manager.

Leicester Xms and Dbs layer are assumed to be in a sepatate data store and are not included here.

7.7 Project Manager Model

The DeltaV System as described here is a combination of the traditional software organization as well as a new service-oriented organization. The traditional organization is structured around technical expertise. This organization typically has control experts, controller experts, IO experts, client-writer experts, database experts, etc. Functional experience is often separate from application development. This style of organization is supported along the lines of deep experience, long learning curves, etc. Many of the principles that support working this way have not gone away – for example control expertise takes many years to develop and is best left with the few folks that completely understand it.

On the other hand service-oriented development is organized around units of delivery. In this model every service, software functional grouping, and application is owned in-total by a specific team. The whole model is optimized for minimal interruption, minimal external control and checks, and minimal waiting on other teams. Such an organization requires that all developers have some level of knowledge about the overall system being built. To keep these functional groups developing there must be a pool of expertise that can deal with specific problems outside of the specific teams functional expertise. Taken to the extreme it should be possible to transfer development activities from one team to another – even if the team is geographically separated from the other teams.

Shown below are runtime and configuration models that will be developed as part of the Starburn project.

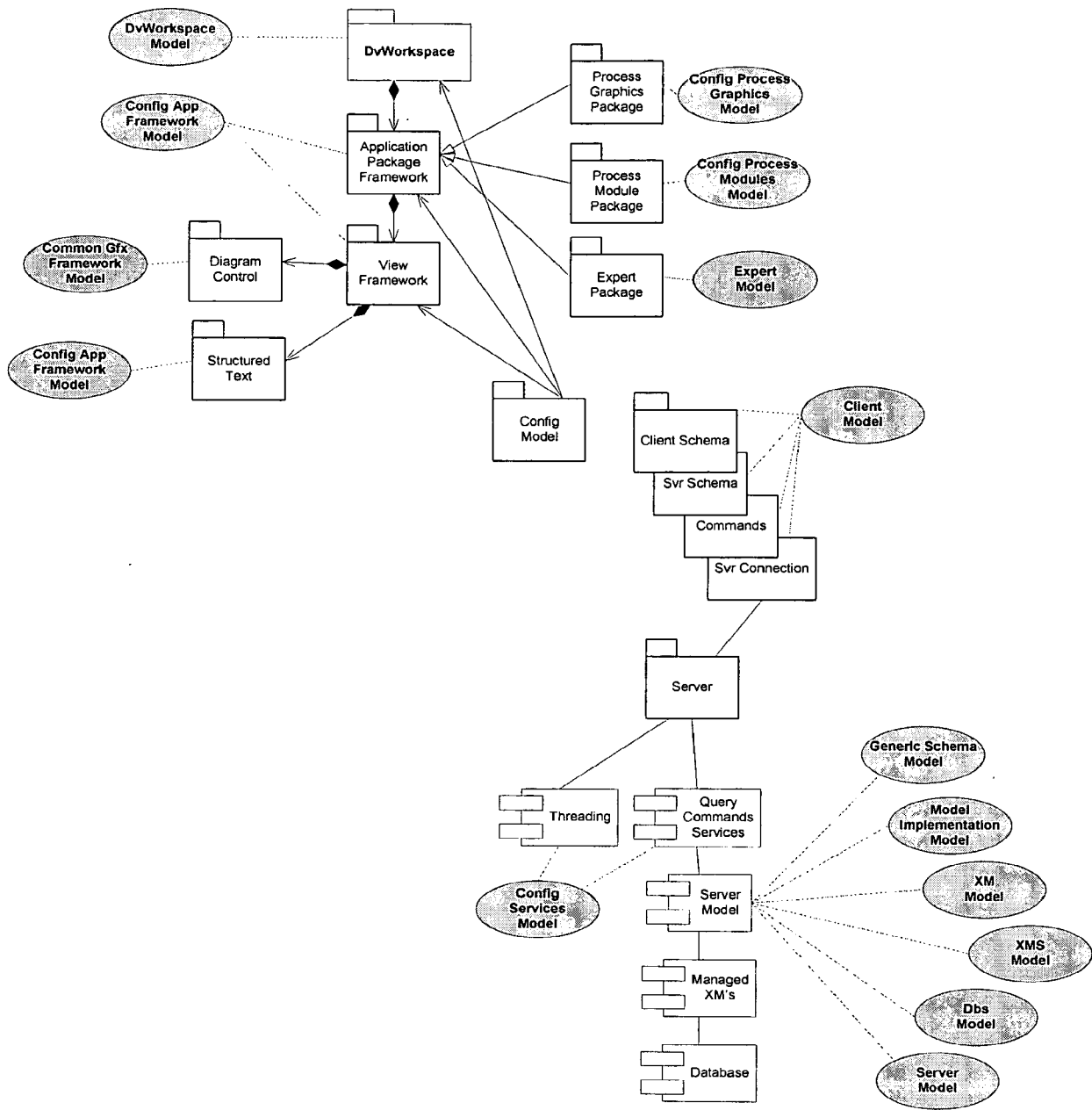


Figure 59. Configuration Models

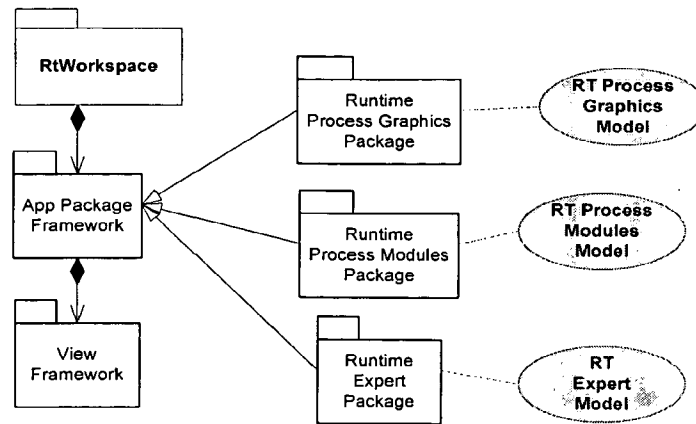


Figure 60. Runtime Models

The diagrams above continue to highlight another important concept presented in this document: the differences between Solution/Application Models and Technical Models. The stereotyping for models fall into the following categories:

- a. **Technical Architecture:** utility components and patterns that cross layers. There are several instances of these including Common, Runtime and Configuration.
- b. **Infrastructure:** the middle layers, which largely facilitate communication and decouple client/server. Instances of this would be e.g. XM, Generic Model, Client Interface, etc.
- c. **Framework:** the class framework and utility components for a particular layer. Instances would be e.g. Client Model Framework, UI Framework.
- d. **Solution/Application:** the collection of components that make up an application. Instances would be e.g. Process Graphics, Process Modules. The runtime and configuration parts of each of these would be separate models as ownership is likely to be in different teams/sites. It is possible that there would be some things shared between configuration and runtime. In that case options would be to put the common things in a separate model or in one of configuration/runtime.
- e. **Component:** a model dedicated to a particular component.

discuss design repositories to some extent here

8 Development Practices

As the saying goes, there are two ways a project can fail, by not following a development process and following a development process⁵⁰. DeltaV has always successfully endorsed a strong development process and will continue to do so. However, we need to extend the process to support the concepts described in this document for versioning and multi-site development.

Our primary objectives in making changes to the development process are to:

- Accommodate a new and rapidly expanding development group in Manila and have this group contribute productively to the Starburn program.
- Enable collaboration with the development group in Pittsburgh.
- Enable the four development sites and the project teams within them to progress parts of the Starburn development in parallel.
- Support the flexibility in versioning and deployment described in the Project Architecture section, thus enabling frequent enhancement of the application layers of the system upon a very stable core.
- Ease the burden of producing and testing localized versions of the software and enable much closer to simultaneous release of base language and localized versions.
- Enhance the quality of our designs.
- Improve the testability of our software.
- <Mark, I'd like to improve the traceability of our software but I don't know if you'd want to put that here.>

Two enhancements to our process in particular stem from advances in software development theory and practice that have emerged since the inception of DeltaV development in 1995. These are

- The introduction of Component and Service Based Development (CBD). This can also be considered an Interface Centric approach to development.
- A conscious focus on Technical Architecture and the separation of it from the Solutions or Applications that provide particular features to the user. Technical Architecture provides common services to the Applications.

Other significant changes are:

- Clear separation of a core system from the software that layers upon it, not only in terms of software artifacts but also in frequency of revision.
- Greater use of automation both in testing, building and documenting the system.

It is worth mentioned some things that we will retain from our previous practices.

- Incremental development, with early increments emphasizing research and prototyping whilst later increments emphasize the structured and formal addition of features to the system. Early increments cut a path through the system from top to bottom with bare bones functionality to establish the architecture and patterns. Subsequent increments broaden the feature set.

⁵⁰ Francois Deverdiere

- The major phases of development we have used before within increments will remain in place, namely Requirements, Analysis, Design, Implementation, Validation and Verification.
- The Iterative approach to development will continue, particularly in the development of non-core parts of the system, so that as before the above phases will, and should run in parallel, with the balance of activity typically shifting towards the “later” phases over time.
- The Object Oriented approach that has served us well will continue to be used within components.
- We will continue to use Select and UML to produce diagrammatic representations of our designs. In Starburn this practice will become more common across the whole development team.
- Direction and Change statements, Use cases and Concept Documents will continue to be the driving forces of requirements.
- We will continue to strive for the re-use of software artifacts wherever possible. However, we will need to strike a balance between re-use and autonomy of components so as not to create unmanageable dependencies. Part of the continue effort on re-use will see us adopting a “pattern” mentality, where developers identify and re-use common patterns in both design and implementation.

Although it is not strictly a process change, it is worth stating here that there will be an increased emphasis on the use of XML throughout the system. XML will form the primary content of many of our interfaces. This keeps the interfaces light in terms of the methods supported, whilst allowing them to be rich in content. The meta-formats of such XML will be defined as part of our design, through schemas (of our own making and also written in XML, but not XSD). Such schemas will allow the collaborators on each side of an interface to validate and interpret the content of messages. XML “blobs” will be used to describe the content of certain items stored in the database. XML will also be used to enable much of our software to be data driven. A particular advantage of using XML in interfaces is that it will better facilitate the capture of data passing through interfaces for the purpose of automated regression testing and diagnostics.

The remainder of this section describes the process changes in more detail and then gives a brief outline of the major phases of development and the artifacts used in them.

At the end of the section are some examples of the analysis and design diagrams we will use. These are samples taken from Select that do not form part of the actual Starburn design.

8.1 Component and Service Based Development

There is a variety of opinion in the literature concerning Component Base Development. We choose to take a pragmatic and readily understandable position. We consider a Component to be an independently deployable unit of functionality, such as a C# assembly, or an executable program.

A component provides Services to other components through well defined Interfaces that provide a contract between the component and the things that use it.

There is a specific design activity in which we identify components and the services they will provide. This can be approached both top down and bottom up. In a top down approach we first identify candidate components that are appropriate to handle particular areas of functionality. The components are then fleshed out by modeling their interiors. In a bottom up approach, first a class model is constructed and then classes are clustered to form components. In practice a combination of the two approaches is likely to be used. Also the design and implementation will be iterated to reach a final, well balanced solution.

Components should be designed to be internally cohesive and externally loosely coupled. There should be no circular dependencies between them.

Some components provide common services to a wide variety of applications within the system, whilst others are more application specific.

An application is typically achieved through the use of many components collaborating to achieve the desired functionality.

Within components we will continue to use object oriented techniques and our implementations will remain object oriented through use of the C# language.

We will continue to use the Select Component Architect OOA/D tool to document Component Based designs, using the UML notation. UML "packages" are used to depict components within the designs. Training in this approach is provided by Select Business Solutions. A UML/Select design guideline will aid the developers in producing good designs. The Select automated Reviewer tool will also be used to help ensure correctness and consistency. Each site will be provided with a copy of this tool, which can be run ad hoc, or as part of the regular build process.

In some cases it may be possible to generate code from the designs, using the Select C# synchronization tool. Each site will be provided with a copy of this tool, which again can be run ad hoc or as part of the build process.

8.1.1 Layering and Stereotyping

Component based designs will make use of layering (tiering) and UML stereotypes to help guide developers in their designs and to provide additional levels of organization and structure. A multi-tiered architecture for the configuration system has already been established during the early increments of Starburn.

8.1.2 Component Management

Having identified components and their interfaces we need to publish information about them to the development teams so that parallel development can proceed with confidence that the interfaces are agreed and will be supported. The Select Component Manager (SCM) provides a means to publish information about components. It supports the Supply, Manage and Consume philosophy (SMAC). Specifications of components can be published from Select Component Architect into SCM. Other designs in SCA can use these specifications. Component implementations can later be added to the SCM repository.

8.2 Architecture Versus Solutions

The Starburn approach more formally recognizes and clarifies what is meant by "Architecture". There is a separation of concerns between Architecture and "Solutions" or applications. Again this is used to guide our designs and development.

Our development will continue to be driven by Use Cases, both for Solutions that have user interfaces and those that do not. As application developers proceed with their designs they will identify Use Cases that apply specifically to their application; these are known as Solution Use Cases. They will also encounter use cases that should not be application specific but should be common across all, or many applications. These will become identified as Architectural Use Cases and will be modeled separately from Solution specific use cases. Features such as checking access rights, exception handling, producing pop-up menus, analyzing reference paths and accessing data sources would all be examples of Architectural Use Cases.

Specific teams and individual developers will be devoted to developing software that will satisfy the needs of the architectural use cases.

The architecture manifests itself as:

- Components that provide common services
- Components that provide helper functions
- Class frameworks from which applications can inherit
- Design and coding patterns that provide guidance as to how developers should best make use of the architectural software.

8.3 Requirements

<Mark, problems I have had with the current artifacts are the fragmented nature of the information, the difficulty of keeping the different artifacts in sync and the lack of clarity over the role within our process of the requirements in each artifact. I would like to more formally define exactly where we state our requirements, such that they can be checked off a) by developers and reviewers to ensure the requirements have been catered for in the design, b) by developers when they are writing test scripts, and c) by the system testers when they are verifying the requirements. You have actually gone some way towards this in your text below.

Perhaps this is a leveled set of information, with the Dir/Change stats providing the first level, concept document providing the second level and Use Cases providing the third level. I am thinking that the Use Cases could be the formal exposition of requirements, though we would have to be sure and document the alternative paths adequately (Tenny has worked on a Use Case template addresses this). We also need to cater for non-feature based requirements such as performance (could be included in the business rules you identified).

Aside: As you probably gathered I like the idea of a requirements database, which can give a lot of power to organize and understand the requirements, structure them hierarchically, track their status, maintain unique Ids, keep an audit trail, enhance traceability, avoid duplication, and enable different views onto the requirements info set. Of course this has its problems, particularly in terms of integration with graphical information, though this is not insurmountable. I don't think we could do it right now but it is a gleam in my eye for when I get further down the road here.>

The following artifacts are produced during the requirements phase for each major feature set to be added to the system. (From a project management point each major feature set gives rise to a Project):

- 1- **A direction statement.** The marketing direction statement contains a feature list, typically a few lines per feature, describing the key aspects of the system. The features are numbered and will act as a check list for both developers and system testers later in the project. Marketing and Technology collaborate to establish the content of the direction statement.
- 2- **A concept document.** The concept document acts as a guiding light for the ensuing development. It includes a rough scope of the system, a description of the main features, typically containing overview diagrams such as network topologies and software layer diagrams, a summary of the key requirements and key technologies (if appropriate), a description of the system's main external interfaces, and in some cases a set of models describing the key features of the system.

The precise scope of the content of a concept document cannot be prescribed for all projects because all will have different characteristics. However, this document is expected to represent the accumulated wisdom on the topic in question and to cover any angles that might come into question during the development. A few examples of the content are why we are doing this, who the target users are, key user visible objects, how downloads need to work and migration from earlier releases.

The key requirements are identified in the concept document by continuous numbered bullets that run throughout the document independently from the section numbering. These key requirements again act as a check list for both developers and testers.

- 3- **An initial user-interface prototype**, where appropriate. This is often the best way to quickly focus the requirements gathering activities while gathering feedback from users, marketing, etc.
- 4- **An initial set of business rules.** This is the set of requirements that the system must satisfy such as constraints upon item naming, and item values, inter-item referencing, performance, compatibility etc. Also included here are requirements concerning aspects of the system such as Version control and Security.

This document persists and is expanded throughout the development cycle, eventually ending up as the definitive statement of the business rules that will govern constraints and validation to be applied in the database, the user interface and the runtime software. It acts as a check list for developers and testers, and a source of information for the documentation team. Extensions to the interface model should be specified <Mark, not sure what you meant by this>.

- 5- **A test specification.** An initial test specification that will be used for acceptance testing, based on the direction statement and concept document.

The set of requirements deliverables will depend on the scope and complexity of the project. In some cases customer involvement is required – in many cases no direct customer involvement is provided until the requirements and concepts have been initially written, intellectual property identified and patented, and the 1st increment of development completed. Based on the initial customer feedback the requirements may have to be adjusted.

The requirements documents for a project will evolve as development increments progress between project initiation and the delivery of a release containing the components produced by the project.

8.4 Analysis

The following artifacts are produced during the analysis phase:

- 1- **User interface Use Case documents.** In these documents a combination of user interface deliverables and activity drawings are developed. The activity drawings should be captured in Select. The use cases will document the normal paths through the software and also alternate paths, which may arise in error situations.
- 2- **System Use Case Documents.** These documents contain use cases that involve no direct interaction with the user, such as how to handle an event such as an alarm within the control software. These use cases should again identify normal and alternate paths.
- 3- **Component diagrams.** These diagrams use UML packages to identify components and how they are related.
- 4- **Class Diagrams.** These may be started at this stage to describe the internals of components or as part of a bottom up approach to component identification.
- 5- **Use Case Diagrams** will be used as the top level representation of use cases and the associated sequence and collaboration diagrams.
- 6- **Component and possibly Object Sequence Diagrams** will be started at this stage.
- 7- **Collaboration diagrams.** These diagrams show how instances of classes or components work together to achieve a particular objective. They help to understand the problems and ensure that component, class and sequence diagrams are correct. They are usually intermediate artifacts that may be discarded.

All the diagrams mentioned above are recorded in Select.

8.5 Design

The following are produced as part of the design phase:

- 1- **The Component Diagrams, Class Diagrams, Use Case Diagrams and Sequence Diagrams** started during Analysis will all be refined and fleshed out at the design stage.
- 2- **State Diagrams, for example describing the life-cycles of objects, will be constructed. These are particularly relevant to runtime software.**
- 3- **Deployment Diagrams.** These diagrams show how components are deployed physically within the system.
- 4- **Detailed design document.** The design document makes use of the detailed select diagrams summarized above. It describes in text the reasoning behind any difficult or contentious design choices and nuances of the design that may not be apparent from the diagrams.

- 5- **Interface specifications.** The interface specification describes the interfaces supplied by and required by each component. The specification includes test specifications for each interface, preconditions & postconditions, expected results, and expected state information. Parameters of the interfaces are detailed. In particular the formats of string parameters that pass Xml structures are specified. The details of the interfaces are recorded in Select Component Architect and published into Select Component Manager.
- 6- **Dependency specification.** The dependency specification includes a defined dependency list and the list of interfaces and methods required. Each dependency may be illustrated by an inter-dependency sequence or collaboration diagram specifying the flow of the processing.
- 7- **Version dependency information.** Using the dependency list and the Dependency Manager the developer should be able to summarize the version dependencies. Together with the project team the combinations of versions that will be supported should be able to be worked out.
- 8- **Download and Import/Export** formats are defined at this stage.

8.6 Implementation

Implementation includes the following:

- 1- The development of the feature set and supporting test scripts. Development will be carried out using Whidbey, the Longhorn version of Microsoft Visual Studio.
- 2- Unit test scripts are written by the developers to cover both the normal and alternate paths of use cases and other documented requirements. The developers ensure that the software builds and the tests run successfully on their local machine before checking both code and test script back into SourceSafe. The Unit tests also act as regression tests.
- 3- Coverage Analysis is carried out to ensure that all code written to implement the required functionality is exercised by the test scripts.
- 4- Integration into the build. All new components that are checked into SourceSafe are incorporated into the daily build process and become available for use by other developers.
- 5- Integration into the test harness. A test harness is provided for developers. This is used to run their tests locally and also to unit/regression tests. When new components are included in the build process the corresponding test scripts are included in the post-build test run.
- 6- System testing. An independent System Test group develops and executes tests based on Direction Statements, Concept Documents and Use Case documents.
- 7- Handoff to test team. Groups of software components are handed off to the System Test team when all requirements have been covered by the implementation, coverage analysis shows that all the software is being exercised by the test scripts and all the tests succeed.

8.7 Validation and Verification

Validation and verification occurs at multiple phases of the project:

- 1- Analysis – developers hold reviews of functional specification, prototypes
- 2- Design – developers hold reviews of models, designs
- 3- Implementation – code and tests scripts are reviewed using walkthroughs or inspections.
- 4- System Test team – the System Test team review their tests to ensure they will adequately validate the required feature set and system as a whole and execute these tests.

At each stage of reviews the various constraints should be discussed.

All reviews are documented in review databases that are maintained at each site.

8.8 Sample Diagrams

8.8.1 Component Diagram

The following Figure is an example of a class diagram showing Components (as UML Packages) and the Interfaces they support. In this case the components are grouped into two layers. Components in the Process Control layer are used to sequence and co-ordinate the activities of the components in the Business Logic layer.

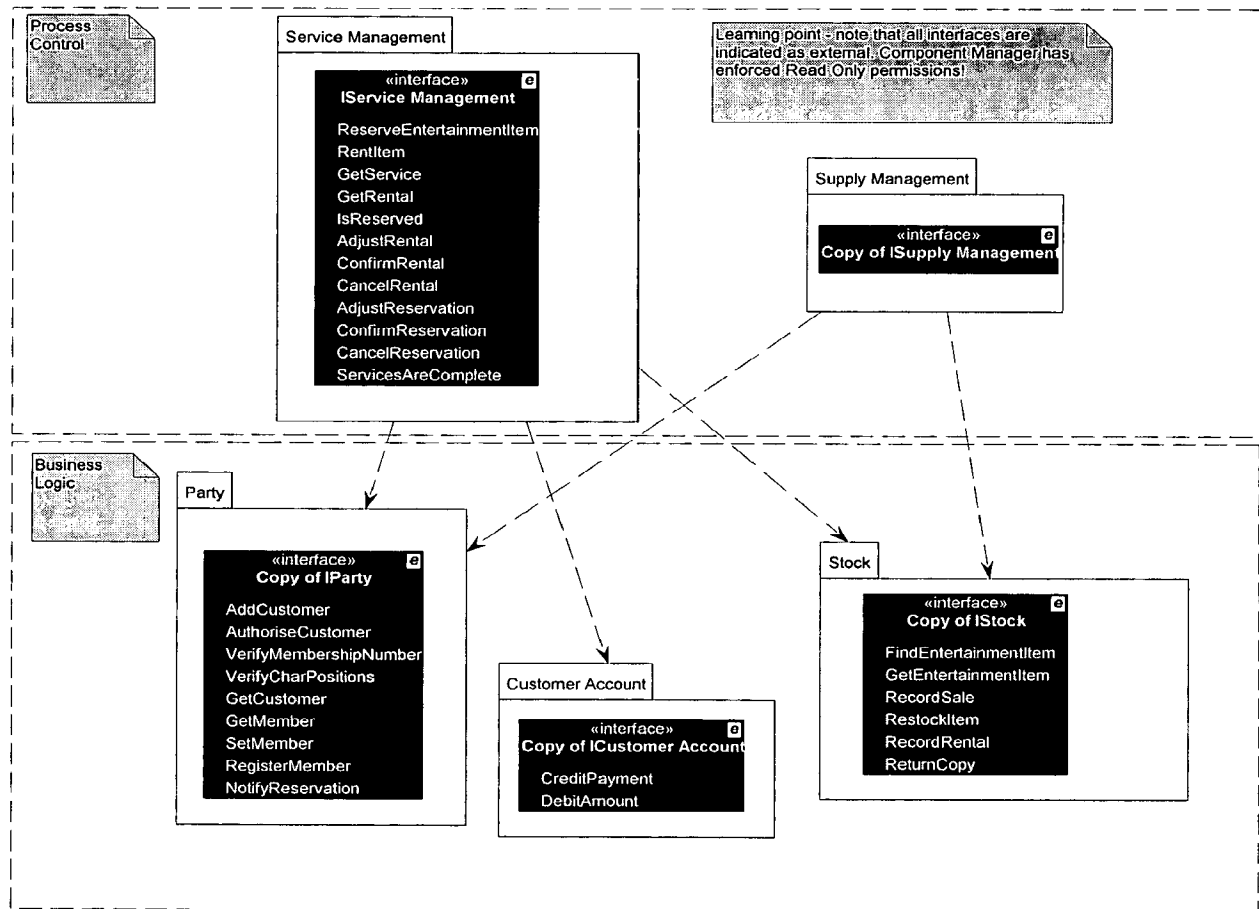


Figure 61 Component Diagram

8.8.2 Component Sequence Diagram

Figure 62 is a sequence diagram showing a sequence of interactions between components.

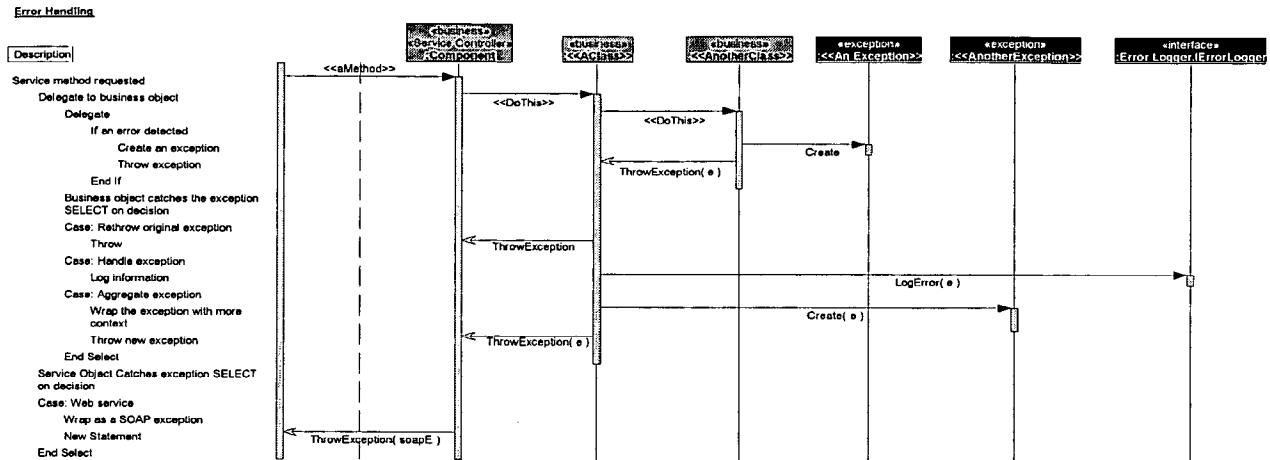


Figure 62 Component Sequence Diagram

8.8.3 Use Case Diagram

Figure 63 is a Use Case diagram showing how external Actors (in this case users) interact with use cases and how use cases are related to each other.

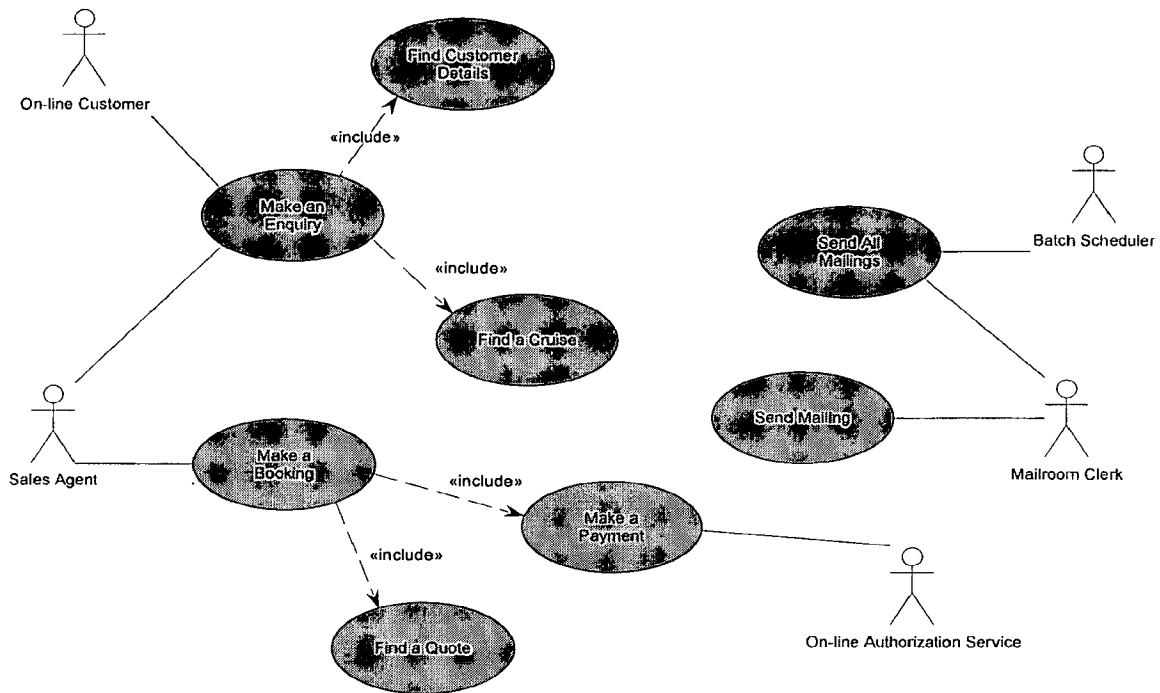


Figure 63 Use Case Diagram

8.8.4 Collaboration Diagram

Figure 64 is a collaboration diagram showing how particular instances of objects and interfaces work together to achieve a specific objective.

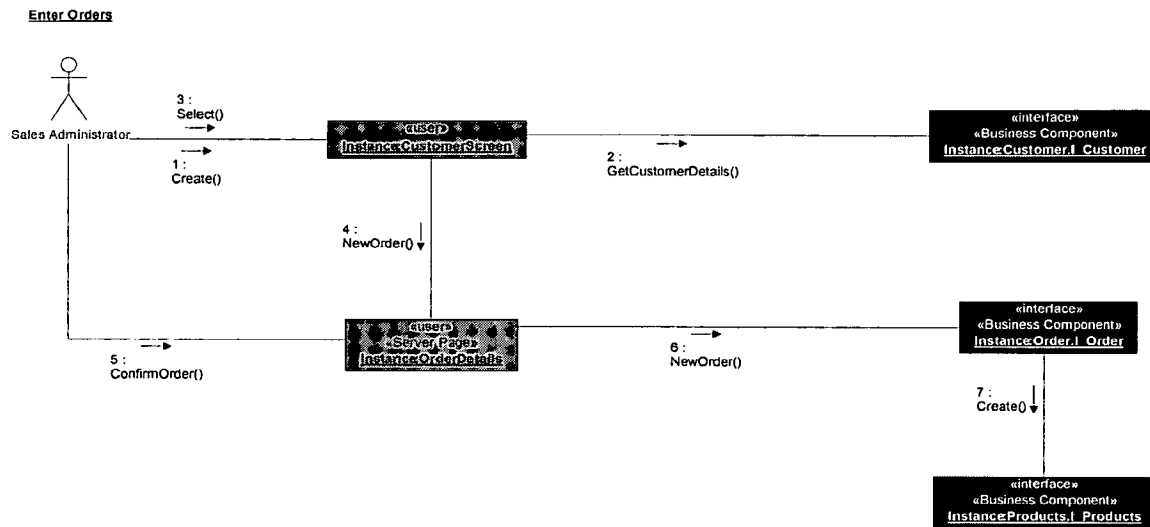


Figure 64 Collaboration Diagram

8.8.5 State Diagram

Figure 65 is a state chart showing the life cycle of an order.

Order

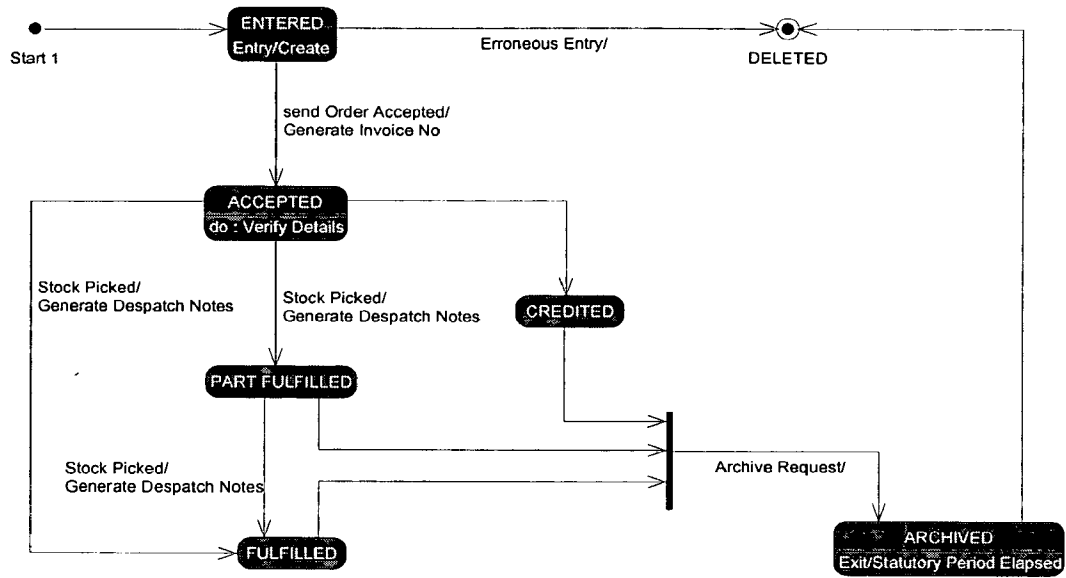


Figure 65 State Diagram

Process Graphics Editor I

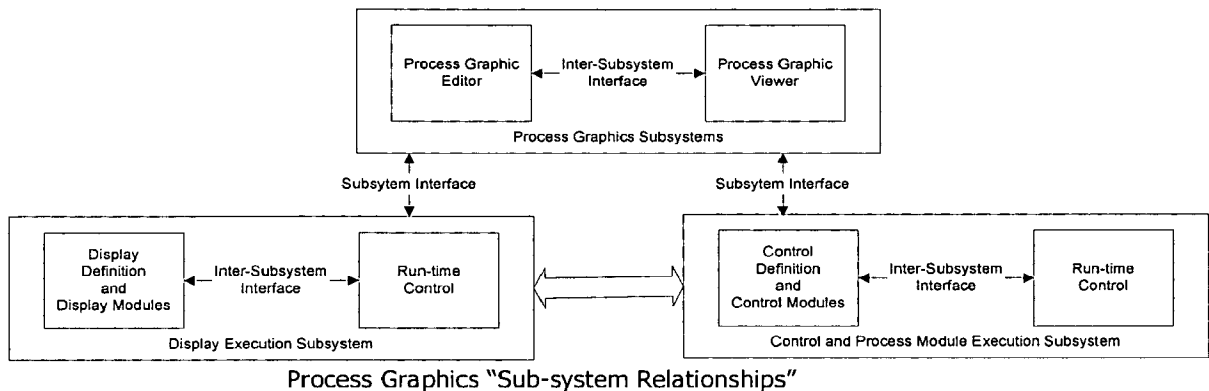
Table of Contents:

1.	Introduction	158
1.1.	Overview of the Process Graphics	158
1.2.	The Process Graphics Editor	158
1.3.	Definitions	159
1.4.	References.....	159
1.5.	Deliverables.....	159
2.	Objectives.....	160
3.	User Interface.....	162
4.	Object Model.....	163
4.1.	Design Diagrams	163
4.1.1.	PG Object Model	163
4.1.2.	PG Editor Forms & Controls	164
4.1.3.	PG Custom Collections & External references	165
4.1.4.	Client Model.....	166
4.2.	Use Cases	169
4.2.1.	Creating a New Display – PUMP	169
4.2.2.	Saving a Display	172
4.2.3.	Loading a previously saved Display	173
4.2.4.	Adding Dynamics to a Display	174
5.	Revision History	177
5.1.	Concept Document.....	177
5.2.	Prototype	177
Figure 1 - Process Graphics Editor screen shot.		159
Figure 2 - PGE UI Components		162
Figure 3 - PG Object Model.....		163
Figure 4 - PGE Forms & Controls		164
Figure 5 - PG Custom Collections & External references.....		165

1. Introduction

1.1. Overview of the Process Graphics

The overall Process Graphics Architecture comprise of Configuration and Runtime Components, which are supported by other subsystems such as Control, Communications, etc. The Configuration System is linked to the runtime through a supporting Runtime workspace and a set of pre-built files providing a set of known behaviors.



1.2. The Process Graphics Editor

As part of the development program, this document describes the concept of the Process Graphics Editor which is one of the many graphical editors available in. Its goal is to identify and define the design implementation for the set of features of the Process Graphics Editor which provides the configuration editor for the operator displays.

This concept document defines the primary objects and functions of the Process Graphics in relation to the Workspace, Client Model, Generic Diagram controls, etc.

The Process Graphics Editor used in the scenarios makes use of a number of features. A snapshot of the editor is included below.

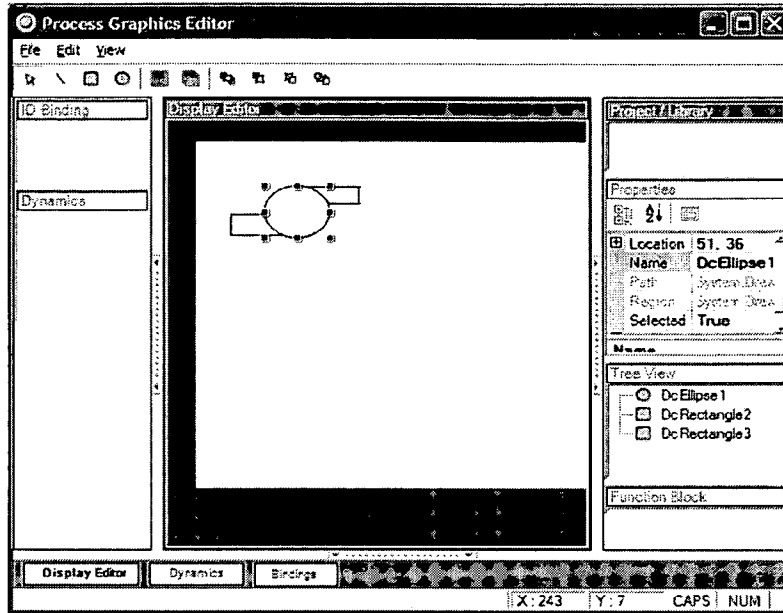


Figure 1 - Process Graphics Editor screen shot.

1.3. Definitions

1.4. References

1.5. Deliverables

2. Objectives

The objective of the Process Graphics Editor is to provide a superior configuration editor for the operator display:

1. The Process Graphics Editor runs on top of the Configuration Workspace.
2. The Process Graphics Editor communicates with the Client Model in retrieving Operator Display data from remote database server.
3. The Process Graphics Editor utilizes the Generic Diagram Control in drawing.
4. Be able to provide a Display Editor view where user can create/modify operator display.
5. Be able to provide a Dynamics Editor where user can create/modify dynamic functionality by use of sequential function blocks.
6. Any static item can be turned into a dynamic item.
7. Be able to provide display elements.
 - a. Shape Elements (Rectangle, Polygon, Circle/Ellipse, Line, Polylines, Path)
 - b. Container elements
 - i. Group
 - ii. SVG
 - iii. Symbols
 - c. Text Elements
 - i. Text, Text path
 - d. Reference Elements
 - i. Image
 - e. Other Elements
 - i. Script
 - ii. PCDDATA
 - iii. description
8. Be able to provide user controls on the display
 - a. Button
 - b. TextBox
 - c. CheckBox
 - d. ListBox
 - e. Slider
9. Be able to add behaviors to the display
 - a. Action, Alert(attach a data mapping)
 - b. Conditions (If, Switch)
 - c. Element (Copy, Create, Move, Replace)
 - d. Listener (register for an event and take action when the event arrives)
 - e. LoadURL
 - f. LoadXML
 - g. Print Element
10. Be able to bind inputs and outputs to different data sources
 - a. DeltaV Runtime
 - b. DeltaV Historian
 - c. OPC
 - d. XML File
 - e. HTTP Link
 - f. Alarm Services
 - g. Event Services
 - h. General Web Services
 - i. Yukon Managed Runtime (ADO interface)
11. Be able to perform the following actions while editing a display
 - a. Copy, Paste, Cut, Delete nodes

- b. Move to Back/Front
 - c. Move One Forward/Backward
- 12. Be able to perform the following actions while editing a dynamic behavior
 - a. Create inputs and outputs
 - b. Add function blocks
 - c. Connect using wires
 - d. Add an IO binding to inputs

3. User Interface

The Process Graphics Editor is composed of the following UI elements:

3.1. Display Editor

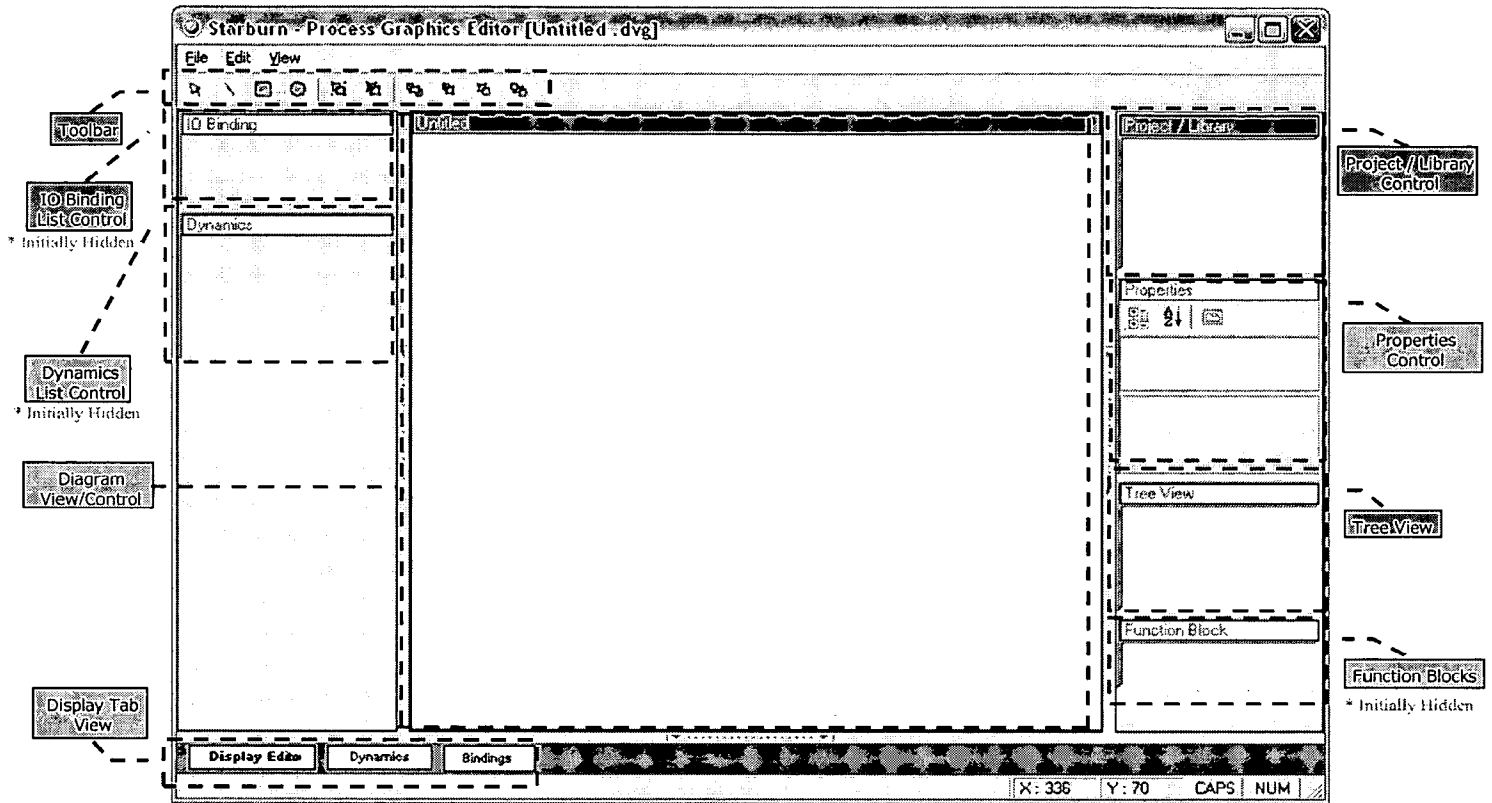


Figure 2 - PGE UI Components

4. Object Model

4.1. Design Diagrams

4.1.1. PG Object Model

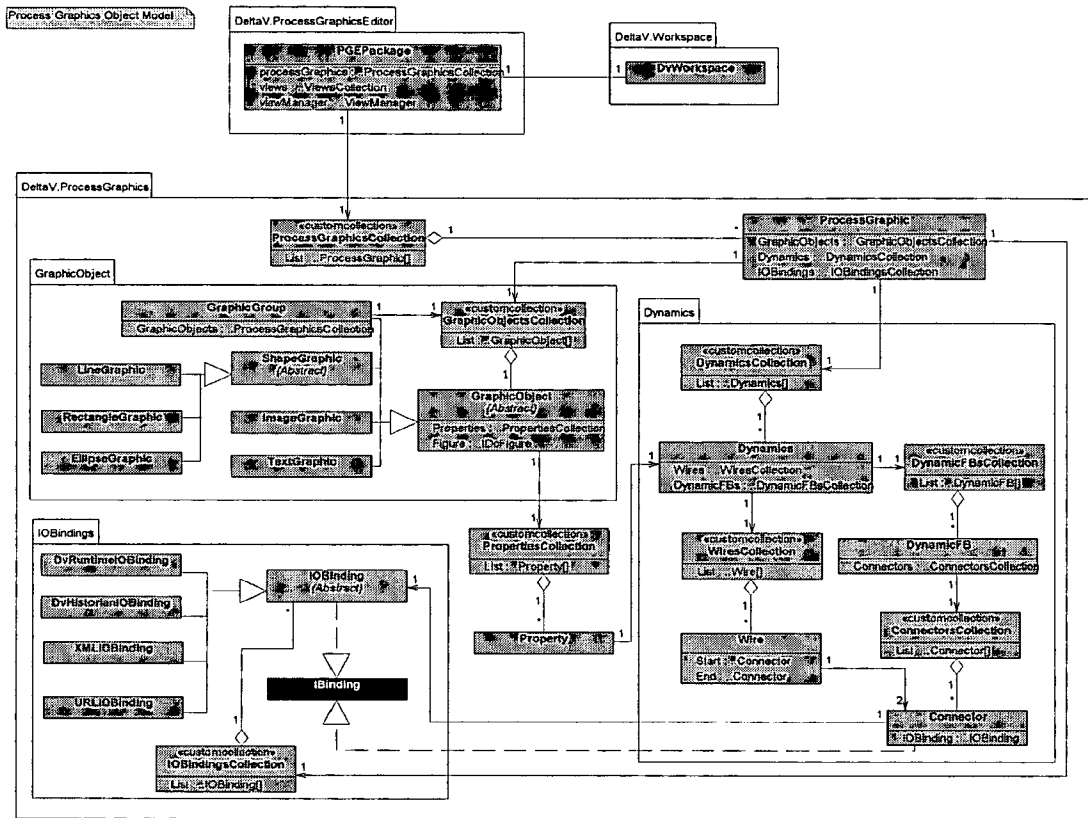


Figure 3 - PG Object Model

4.1.2. PG Editor Forms & Controls

All forms/views and controls are derived from Configuration Workspace components.

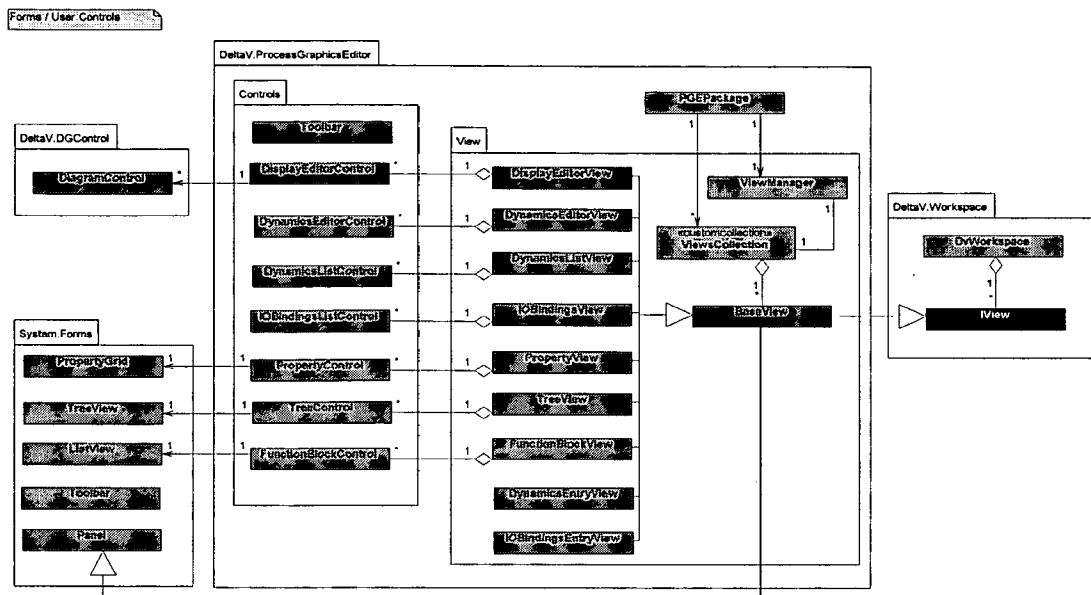


Figure 4 - PGE Forms & Controls

4.1.3. PG Custom Collections & External references

PG Custom Collections are derived from System.Collections.CollectionBase to easily support .NET data bound controls.

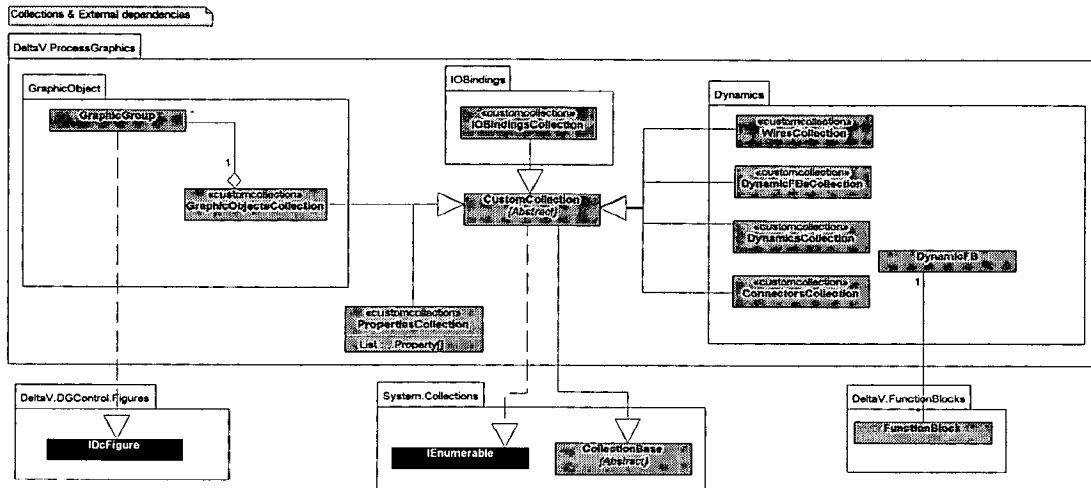


Figure 5 - PG Custom Collections & External references

4.1.4. Client Model

The Display definition will be stored in the database as an xml, wherein the details of the display are stored as an XML blob.

Listed below is an example of what the Client Schema may look like.

```
<?xml version="1.0" encoding="utf-8" ?>
<Client>
  <Type name="Root" serverType="Site">
    <Properties allServerProperties="F" />
    <Roles allServerRoles="F">
      <Role name="ProcessGraphics" serverRoleMapping="ProcessGraphics" destType="ProcessGraphic" />
    </Roles>
  </Type>
  <Type name="ProcessGraphic" serverType="ProcessGraphic">
    <Properties allServerProperties="F">
      <Property name="name" dataType="String" serverPropertyMapping="(name)" />
      <Property name="filename" dataType="String" serverPropertyMapping="(filename)" />
      <Property name="width" dataType="Int32" serverPropertyMapping="(width)" />
      <Property name="height" dataType="Int32" serverPropertyMapping="(height)" />
      <Property name="content" dataType="XML" serverPropertyMapping="(xml blob)" />
    </Properties>
  </Type>
</Client>
```

Listed below is an example of the content of an XML blob. It may conform with the client model so that it is portable with the Client Model components.

```
<?xml version="1.0" encoding="utf-8" ?>
<Client>
  <Type name="Root" serverType="Site">
    <Properties allServerProperties="F" />
    <Roles allServerRoles="F">
      <Role name="ProcessGraphics" serverRoleMapping="ProcessGraphics" destType="ProcessGraphic" />
    </Roles>
  </Type>
  <Type name="ProcessGraphic" serverType="ProcessGraphic">
    <Properties allServerProperties="F">
      <Property name="name" dataType="String" serverPropertyMapping="(name)" />
      <Property name="filename" dataType="String" serverPropertyMapping="(filename)" />
    </Properties>
    <Roles allServerRoles="F">
      <Role name="Groups" serverRoleMapping="Groups" destType="Group" />
      <Role name="Elements" serverRoleMapping="Elements" destType="Element" />
      <Role name="Dynamics" serverRoleMapping="Dynamics" destType="Dynamic" />
      <Role name="IOBindings" serverRoleMapping="IOBindings" destType="IOBinding" />
    </Roles>
  </Type>
  <Type name="Group" serverType="Group">
    <Properties allServerProperties="F">
      <Property name="name" dataType="String" serverPropertyMapping="(name)" />
      <Property name="x" dataType="Int32" serverPropertyMapping="(x)" />
      <Property name="y" dataType="Int32" serverPropertyMapping="(y)" />
      <Property name="width" dataType="Int32" serverPropertyMapping="(width)" />
      <Property name="height" dataType="Int32" serverPropertyMapping="(height)" />
      <Property name="backgroundcolor" dataType="Int32" serverPropertyMapping="(backgroundcolor)" />
      <Property name="foregroundcolor" dataType="Int32" serverPropertyMapping="(foregroundcolor)" />
      <Property name="fillstyle" dataType="String" serverPropertyMapping="(fillstyle)" />
    </Properties>
    <Roles allServerRoles="F">
      <Role name="ProcessGraphics" serverRoleMapping="ProcessGraphics" destType="ProcessGraphic" />
    </Roles>
  </Type>
</Client>
```

```

<Type name="Element" serverType="Element">
  <Properties allServerProperties="F">
    <Property name="name" dataType="String" serverPropertyMapping="(name)" />
    <Property name="backgroundcolor" dataType="Int32" serverPropertyMapping="(backgroundcolor)" />
    <Property name="x" dataType="Int32" serverPropertyMapping="(x)" />
    <Property name="y" dataType="Int32" serverPropertyMapping="(y)" />
    <Property name="width" dataType="Int32" serverPropertyMapping="(width)" />
    <Property name="height" dataType="Int32" serverPropertyMapping="(height)" />
  </Properties>
  <Roles allServerRoles="F">
    <Role name="Polygons" serverRoleMapping="Polygons" destType="Polygon" />
    <Role name="Ellipses" serverRoleMapping="Ellipses" destType="Ellipse" />
    <Role name="Rectangles" serverRoleMapping="Rectangles" destType="Rectangle" />
    <Role name="Lines" serverRoleMapping="Lines" destType="Line" />
    <Role name="Texts" serverRoleMapping="Texts" destType="Text" />
    <Role name="Images" serverRoleMapping="Images" destType="Image" />
  </Roles>
</Type>

<Type name="Polygon" serverType="Polygon">
  <Properties allServerProperties="F">
    <Property name="name" dataType="String" serverPropertyMapping="(name)" />
    <Property name="foregroundcolor" dataType="Int32" serverPropertyMapping="(foregroundcolor)" />
    <Property name="fillstyle" dataType="String" serverPropertyMapping="(fillstyle)" />
  </Properties>
  <Roles allServerRoles="F">
  </Roles>
</Type>

<Type name="Ellipse" serverType="Ellipse">
  <Properties allServerProperties="F">
    <Property name="name" dataType="String" serverPropertyMapping="(name)" />
    <Property name="foregroundcolor" dataType="Int32" serverPropertyMapping="(foregroundcolor)" />
    <Property name="fillstyle" dataType="String" serverPropertyMapping="(fillstyle)" />
  </Properties>
  <Roles allServerRoles="F">
  </Roles>
</Type>

<Type name="Rectangle" serverType="Rectangle">
  <Properties allServerProperties="F">
    <Property name="name" dataType="String" serverPropertyMapping="(name)" />
    <Property name="foregroundcolor" dataType="Int32" serverPropertyMapping="(foregroundcolor)" />
    <Property name="fillstyle" dataType="String" serverPropertyMapping="(fillstyle)" />
  </Properties>
  <Roles allServerRoles="F">
  </Roles>
</Type>

<Type name="Line" serverType="Line">
  <Properties allServerProperties="F">
    <Property name="name" dataType="String" serverPropertyMapping="(name)" />
    <Property name="startx" dataType="String" serverPropertyMapping="(startx)" />
    <Property name="starty" dataType="String" serverPropertyMapping="(starty)" />
    <Property name="endx" dataType="String" serverPropertyMapping="(endx)" />
    <Property name="endy" dataType="String" serverPropertyMapping="(endy)" />
  </Properties>
  <Roles allServerRoles="F">
  </Roles>
</Type>

<Type name="Text" serverType="Text">
  <Properties allServerProperties="F">
    <Property name="name" dataType="String" serverPropertyMapping="(name)" />
    <Property name="fontname" dataType="String" serverPropertyMapping="(fontname)" />

```

```

<Type name="Image" serverType="Image">
  <Properties allServerProperties="F">
    <Property name="name" dataType="String" serverPropertyMapping="(name)" />
    <Property name="location" dataType="String" serverPropertyMapping="(location)" />
    <Property name="backgroundstyle" dataType="String" serverPropertyMapping="(backgroundstyle)" />
  </Properties>
  <Roles allServerRoles="F">
  </Roles>
</Type>
<Type name="Dynamic" serverType="Dynamic">
  <Properties allServerProperties="F">
    <Property name="name" dataType="String" serverPropertyMapping="(name)" />
  </Properties>
  <Roles allServerRoles="F">
    <Role name="FBDynamics" serverRoleMapping="FBDynamics" destType="FBDynamic" />
  </Roles>
</Type>
<Type name="FBDynamic" serverType="FBDynamic">
  <Properties allServerProperties="F">
    <Property name="name" dataType="String" serverPropertyMapping="(name)" />
  </Properties>
  <Roles allServerRoles="F">
    <Role name="FBBlocks" serverRoleMapping="FBBlocks" destType="FBlock" />
    <Role name="Wires" serverRoleMapping="Wires" destType="Wire" />
  </Roles>
</Type>
<Type name="FBlock" serverType="FBlock">
  <Properties allServerProperties="F">
    <Property name="name" dataType="String" serverPropertyMapping="(name)" />
  </Properties>
  <Roles allServerRoles="F">
    <Role name="Connectors" serverRoleMapping="Connectors" destType="Connector" />
  </Roles>
</Type>
<Type name="Connector" serverType="Connector">
  <Properties allServerProperties="F">
    <Property name="name" dataType="String" serverPropertyMapping="(name)" />
  </Properties>
  <Roles allServerRoles="F">
    <Role name="ConnectionTypes" serverRoleMapping="ConnectionTypes" destType="ConnectionType" />
  </Roles>
</Type>
<Type name="ConnectionType" serverType="ConnectionType">
  <Properties allServerProperties="F">
    <Property name="name" dataType="String" serverPropertyMapping="(name)" />
  </Properties>
  <Roles allServerRoles="F">
    <Role name="Inputs" serverRoleMapping="Inputs" destType="Input" />
    <Role name="Outputs" serverRoleMapping="Outputs" destType="Output" />
  </Roles>
</Type>
<Type name="Input" serverType="Input">
  <Properties allServerProperties="F">
    <Property name="name" dataType="String" serverPropertyMapping="(name)" />
    <Property name="type" dataType="String" serverPropertyMapping="(type)" />
    <Property name="default" dataType="String" serverPropertyMapping="(default)" />
  </Properties>
  <Roles allServerRoles="F">
  </Roles>
</Type>

```

```

<Type name="Output" serverType="Output">
  <Properties allServerProperties="F">
    <Property name="name" dataType="String" serverPropertyMapping="(name)" />
    <Property name="type" dataType="String" serverPropertyMapping="(type)" />
    <Property name="default" dataType="String" serverPropertyMapping="(default)" />
  </Properties>
  <Roles allServerRoles="F">
  </Roles>
</Type>
<Type name="Wire" serverType="Wire">
  <Properties allServerProperties="F">
    <Property name="name" dataType="String" serverPropertyMapping="(name)" />
  </Properties>
  <Roles allServerRoles="F">
    <Role name="Connectors" serverRoleMapping="Connectors" destType="Connector" />
  </Roles>
</Type>
<Type name="XMLTag" serverType="XMLTag">
  <Properties allServerProperties="F">
    <Property name="name" dataType="String" serverPropertyMapping="(name)" />
  </Properties>
  <Roles allServerRoles="F">
  </Roles>
</Type>
<Type name="OPCTag" serverType="OPCTag">
  <Properties allServerProperties="F">
    <Property name="name" dataType="String" serverPropertyMapping="(name)" />
  </Properties>
  <Roles allServerRoles="F">
  </Roles>
</Type>
<Type name="DVRuntimeTag" serverType="DVRuntimeTag">
  <Properties allServerProperties="F">
    <Property name="name" dataType="String" serverPropertyMapping="(name)" />
    <Property name="path" dataType="String" serverPropertyMapping="(path)" />
  </Properties>
  <Roles allServerRoles="F">
  </Roles>
</Type>
<Type name="DVHistorianTag" serverType="DVHistorianTag">
  <Properties allServerProperties="F">
    <Property name="name" dataType="String" serverPropertyMapping="(name)" />
  </Properties>
  <Roles allServerRoles="F">
  </Roles>
</Type>
</Client>

```

4.2. Use Cases

4.2.1. Drawing a New Display – PUMP

4.2.1.1. Pre-Conditions

- a) Process Graphics Package has already been loaded on to the Configuration Workspace.
- b) A new display has already been created.

4.2.1.2. User Action/System Response

User Actions	System Response
User selects circle tool on the PGE display toolbar.	PGE package informs the Generic Diagram Control that the circle tool is active.
User clicks, drags and releases mouse button on the diagram control area to specify circle location and size.	Diagram Control draws a circle.
	A graphic object is added to the display model representing the circle.
	Tree view is refreshed.
	Property view is refreshed.
User selects line or rectangle	System process similar to drawing a circle.
User selects figures on diagram control.	Diagram control adds surrounding objects into a selected figures list and fires an event.
User clicks on grouping tool on the PGE display toolbar.	PGE package informs Display Editor view to group selected figures.
	Display Editor creates a new composite figure from the selected figures and invokes an addFigure() on Diagram control.
	Diagram control refreshes group display.
	Display Editor sets the current context of PGE Package to the newly created group.
	Tree view is refreshed.
	Property view is refreshed.

4.2.1.3. Object Collaboration Diagram

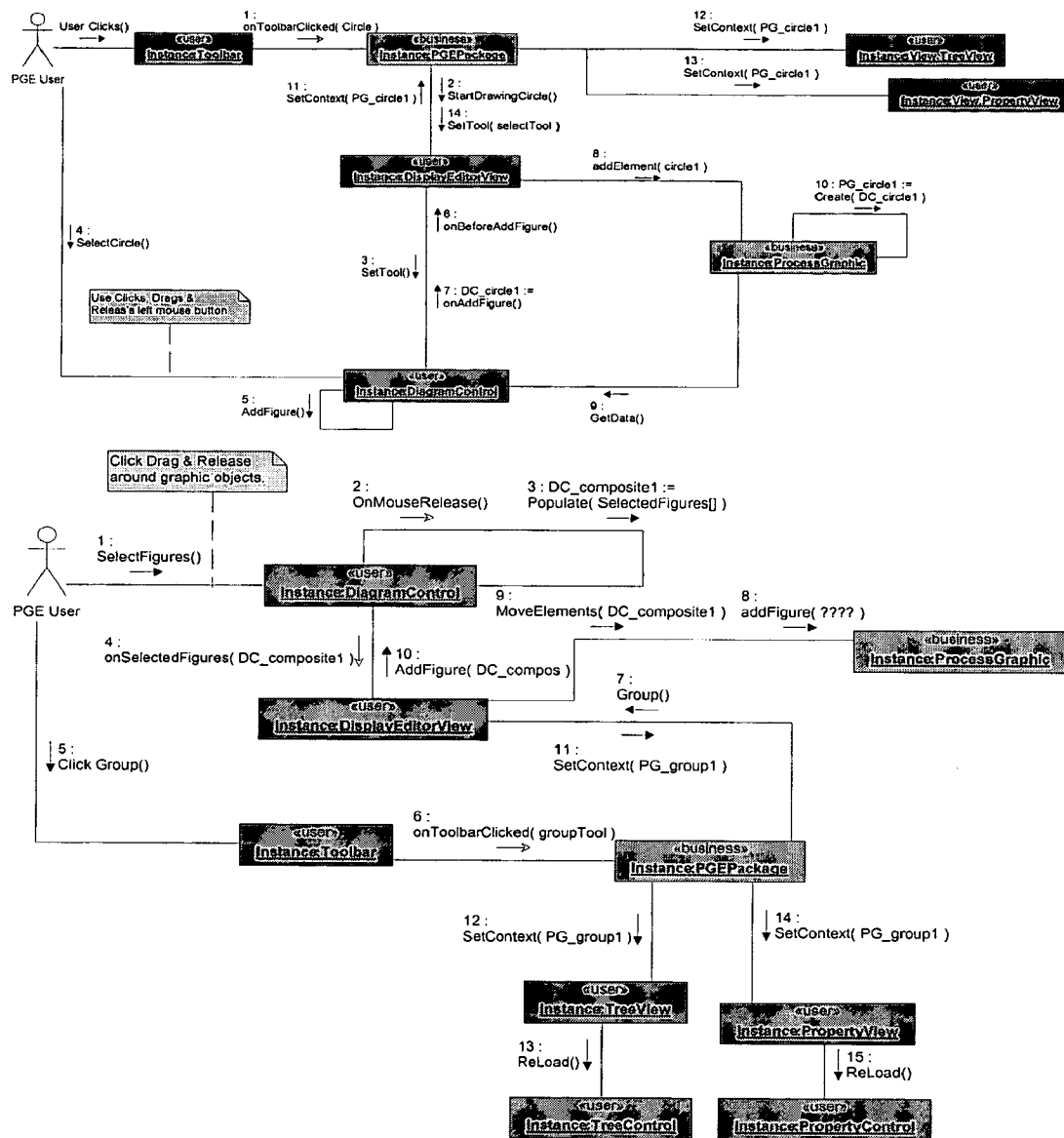


Figure 6 – Drawing a new display (OCD)

4.2.1.4. Post-Condition

- a) A grouped graphic elements representing a pump is now displayed in the display editor.

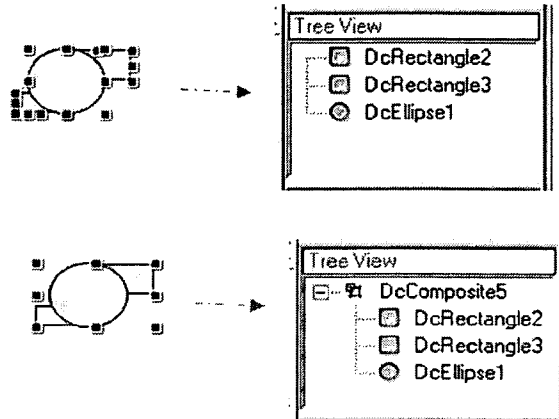


Figure 7 – Draw a pump, grouped output.

4.2.2. Saving a Display

4.2.2.1. Pre-Conditions

- a) Process Graphics Package has already been loaded on to the Configuration Workspace.
- b) A previously loaded or created display is active.
- c) The user is in edit mode.

4.2.2.2. User Action/System Response

User Actions	System Response
User clicks File Save from the workspace menu.	Ask for confirmation
	Update the PG object model
	Converts and saves the PG display as an XML blob
	Update client model

4.2.2.3. Object Collaboration Diagram



Figure 8 - Saving a Display (OCD)

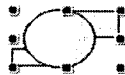
4.2.2.4. Post-Condition

- a) Display is saved on the database via the client model
- b) Is stored as XML blob.

4.2.3. Loading a previously saved Display

4.2.3.1. Pre-Conditions

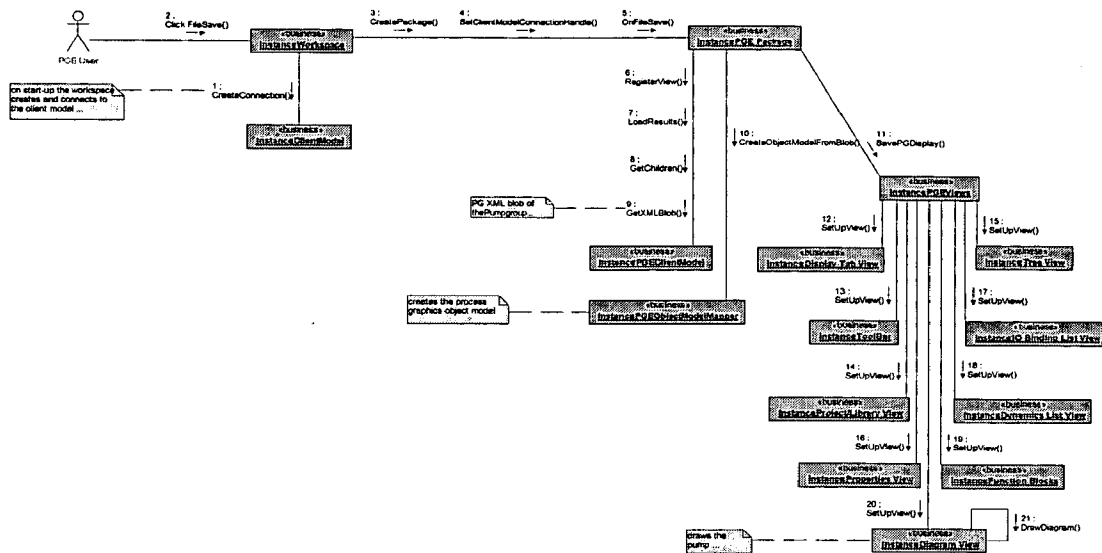
- a) Process Graphics Package has already been loaded on to the Configuration Workspace.
- b) A previously saved display (ex. A group pump)
- c) Note for this scenario, a pump group is consists of other drawing elements :
A circle and 2 rectangles



4.2.3.2. User Action/System Response

User Actions	System Response
User selects "Pump" from the File Open Dialog box in the workspace	Retrieve PG XML blob from the client model
	Map the PG XML blob to PG actual object model
	Set-up the proper PGE views
	A pump group is drawn in the diagram view

4.2.3.3. Object Collaboration Diagram



4.2.4.3. Object Collaboration Diagram

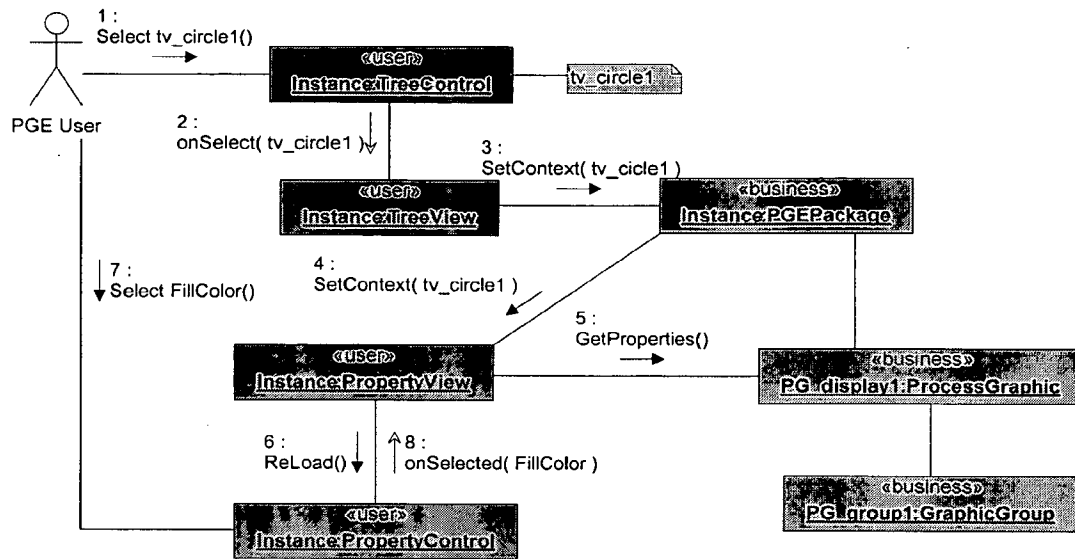


Figure 10 - Adding Dynamics - A (OCD)

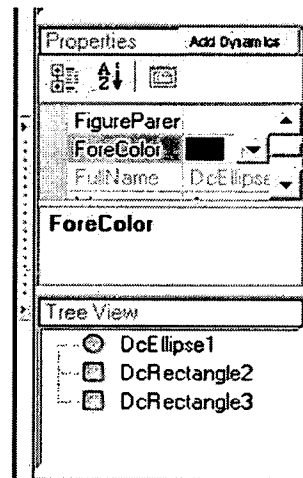


Figure 11 - Adding Dynamics – A (Screen Output)

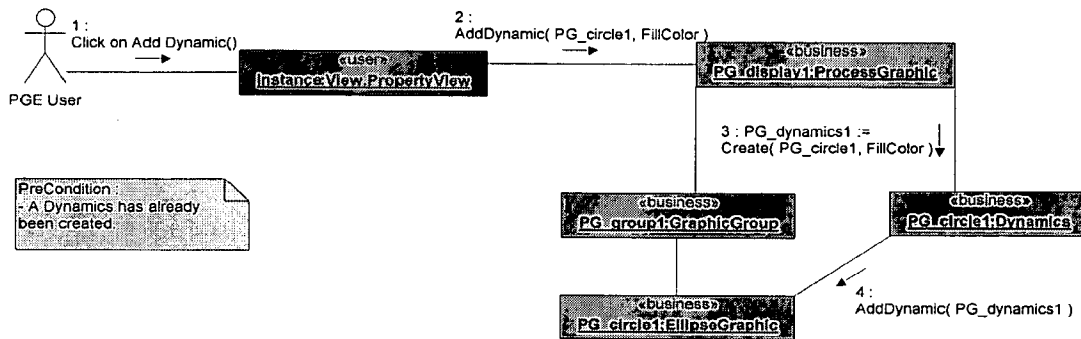


Figure 12 - Adding Dynamics – B (OCD)

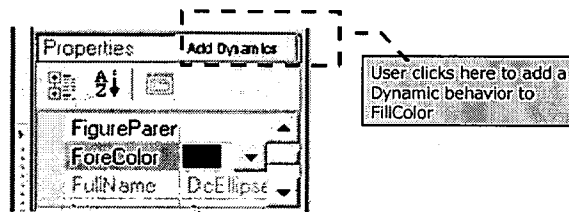
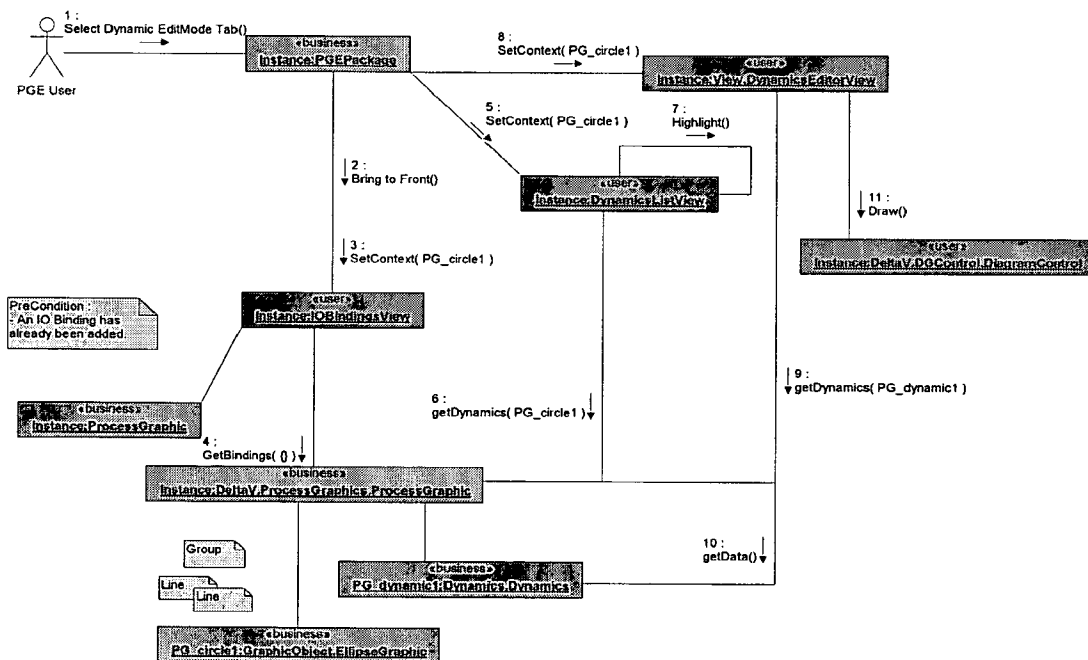


Figure 13 - Adding Dynamics - B (Screen Output)



4.2.4.4. Post - Conditions

A dynamic behavior on Fill-Color has been added to the circle element.



- On/Off
- Changes Color

Process Graphics Editor II

Table of Contents:

1	Introduction	181
1.1	Objectives	181
1.2	Objectives	181
1.3	Overview	181
1.3.1	Smart Process Graphics Studio	182
1.3.2	Process Graphics Server	183
1.4	In this concept document	183
1.5	The Process Graphics Editor	184
1.5.1	184
1.6	The Process Graphics Runtime	184
1.6.1	184
1.7	References	184
2	Process Graphics Display Editor Requirements	184
2.1	General Requirements	184
2.2	Graphic Primitives	184
2.2.1	UI Controls	184
2.2.2	Behaviors	185
2.2.3	Attributes	185
2.2.4	Elements	185
2.2.5	Commands	185
2.3	Parameter References	185
2.4	Binding to data sources	186
2.4.1	Create Input / Output	186
2.4.2	Define Functionality of Dynamic Elements	186
2.4.3	Binding Input / Outputs to Functions	186
2.5	Projects	186
2.6	Displays	186
2.7	Groups	186
2.8	Composites	186
2.9	Faceplates	186
2.10	Dynamos	186
2.11	Data Binding Sources	186
2.11.1	DeltaV Runtime	186
2.11.2	DeltaV Historian	186
2.11.3	DeltaV Alarms	186
2.11.4	DeltaV Events	186
2.11.5	OPC	186
2.11.6	XML File	186
2.11.7	HTTP Browser Link	186
2.11.8	General Web Services	186
2.11.9	Yukon Managed Runtime via ADO	186
2.11.10	ODBC	186
2.12	Scripting	187
2.13	Display Format	187
2.13.1	XMAL format	187
2.13.2	Canvas Size	187
2.13.3	References	187
2.14	Controls	187
2.14.1	Real-time Trends	187
2.14.2	Historical Plots	187
2.14.3	Charts	187
2.14.4	Alarm Banner	187
2.14.5	Alarm Summary	187
2.14.6	Advanced Control	187
2.14.7	Batch	187
2.15	Tag Group Editor	187

2.16	Multimedia.....	187
2.16.1	Sound	187
2.16.2	Video	188
2.17	System Preferences	188
2.17.1	Color	188
2.17.2	Fonts.....	188
2.17.3	Locale	188
2.18	Reports	188
2.19	Diagnostics	188
2.20	Performance Checking Displays	188
2.21	Experts.....	188
3	Process Graphics Display Requirements	188
3.1	General Requirements	188
4	Platform Requirements	188
4.1	Tablet	188
4.2	Multi-screen Configurations	188
4.3	Handhelds	188
5	Process Graphics Runtime Requirements.....	188
5.1	Loading a Display / Project at Runtime.....	188
5.2	Preloading Displays	188
5.3	Display Navigation	188
5.4	Custom Keyboard	188
5.5	Multi-screen Monitor.....	188
6	Scenarios	189
6.1	User Creates Display	189
6.2	Building a Display.....	189
6.3	Building a Composite	189
6.4	Working with Controls	189
7	Process Graphic Design	189
7.1	Configuration Workspace.....	189
7.1.1	Multi-document Interface	189
7.1.2	Projects.....	189
7.2	Displays.....	189
7.2.1	Format	189
7.2.2	Linking	189
7.3	Menus and Toolbars	189
7.3.1	Menus.....	189
7.3.2	Toolbars.....	189
7.4	Views.....	189
7.5	Controls.....	189
7.6	Interface with Configuration	189
7.7	Interface with Runtime	189
7.8	Display Quickview	189
7.9	Project Files	189
7.10	190

1 Introduction

1.1 Objectives

1.2 Objectives

The objectives for the Smart Process Graphic Display project are:

- 1- Integrated – Process Graphic Displays are fully integrated with the configuration and runtime subsystems.
- 2- Quick Edit/Runtime – Users will be able to quickly switch to a run-mode to see how the display they are working on will look at run-time. Both configuration and runtime views can be run simultaneously.
- 3- Ease of use – The Process Graphics System will be easy to use. Displays will be easy to call up on a wide variety of display devices. No display conversion is required to use displays on different devices. It will be easy to design displays for optimum usability on specific form factors.
- 4- Best engineering practices – The system will be designed to facilitate maximum re-use of displays and sub-displays through both standard and user-defined libraries, display classes, etc, thus minimizing configuration and testing effort.
- 5- Separation of the graphics from the scripting engine. [For example, the graphics and it dynamics are defined in data (xml) that is interpreted by scripts (C#).]
- 6- Flexible data bindings – the Process Graphic Displays are able retrieve data from multiple data sources including DeltaV and Ovation, OPC, Web Services, XML Files, etc.
- 7- Smart Objects – The Process Graphic Displays will fully support smart objects.
- 8- Longhorn – Designed from the ground up to take advantage of Longhorn Technologies and Look/Feel.
- 9- Wide range of devices – Runs on a wide range of devices – all the way from dedicated smart phones up through multi-screen consoles.

1.3 Overview

Smart Process Graphic Displays are the center piece of the Starburn release. The display format makes use of Microsoft's XMAL format. The XMAL format specifies the properties of drawing elements for vector graphics commands. An XMAL script can include static data as well as dynamic data. The dynamic elements and attributes can be associated with real-time values, historical values, physical properties such as mass-flow and composition, etc. In this sense XMAL is data driven.

XMAL vector images are stored as text-based commands. Since it is vector based, it is possible to have the same high quality wherever a display is rendered - whether on smart phones, handhelds, or high-end monitors.

Since XMAL is entirely text based it is convenient for DeltaV to store and emit as a download scripts. Using the same techniques used today it is also possible to generate download status indicators, participate in version control and upgrades, be managed as part of Class-based configurations, etc. Process Graphics are downloaded in much the same way that IO and Control, System Setup data, Batch, Advanced Control, and SIS configuration scripts are

downloaded today. Since Process Graphics are stored as text, it will also be convenient for users to search for text within images.

At its core XMAL supports vector graphic shapes consisting of straight lines and curves, images, and text. It supports the basic graphical shapes including rectangles, circles, ellipses, and polygons. It also gives developers the ability to control and implement animation techniques that can be as complicated as generating three-dimensional morphing effects to simple two-dimensional images. All the graphical information is stored in a sequence of commands that draw various vector graphic shapes. This information can be converted by various methods to display an application-specific graphics image. Scalability is achieved without jagged edges because redrawing instructions are sent to the rendering program, rather than sent as pixel values in a bitmap. Rendering engines are defined for each target environment.

Every XMAL element and every attribute of an element in a Process Graphic Display can be animated. It supports bitmap-style filter effects for creating high-impact graphics. XMAL's filter effects give developers the ability to add effects directly to shapes and text in a manner similar to those achieved using professional imaging tools. XMAL has the support for blending, tiling, transforming or rotating elements, morphing, offset, merge, and special lighting effects. Filter effects can be applied separately or in any combination on a vector image. Special effects such as mouse over features can be assigned to any XMAL graphical object.

XMAL produces small file sizes that allow for fast downloading. Additionally, since DeltaV will include both Groups and Composites, it is possible to make a change to a single composite and affect many displays¹.

XMAL can be drawn using data at run time that is derived from a number of different sources in a distributed computing environment. The data can come from DeltaV Runtime, DeltaV Historian, ODBC, Web Services, an XML file, or many other sources. To accomplish this an abstraction or a logic layer is created to establish the relationship of the data to the XMAL graphic. For instance, suppose XMAL describes an image that represents data in terms of pixels (e.g. filling a bar). The real-time data may be stored in units of degrees Fahrenheit. It would be necessary to establish some type of logical relationship between the image and the data that will be displayed in that image so the end user can view something in a format that is easily understood and not subject to misinterpretation. Build-in support from DeltaV will provide the scaling before the data value is rendered.

The overall Process Graphics functionality is realized by two major applications. The configuration environment consists of Smart Process Graphics Studio. Smart Process Graphics Studio includes several tools that are used for creating graphics and graphic components and binding graphics to data sources. The second application is the Smart Process Graphics Server. The server provides back-end support for executing Smart Process Graphic Displays.

1.3.1 Smart Process Graphics Studio

Smart Process Graphics Studio is a visual development workspace designed to create and test Process Graphics. Smart Process Graphics Studio lets developers use an integrated set of tools to build Process Graphic Displays with a separation between the graphical, functional, and data components of displays. This ability to separate the different components results in display formats that are more resistant to breaking when changes are made to the data or whenever graphic changes are made. Separating out the data bindings also makes it easier to bind references to the rest of the system (DeltaV and Ovation), track changes, version displays, etc.

The Smart Process Graphics Editor includes the following:

¹ It is also possible to not effect more than one display by converting the composite from a linked to an embedded composite – likely to be the default behavior.

- Display Editor
- Template Builder
- Data Mapper

1.3.1.1 Display Editor

The Display Builder provides the tools and features required to create rich Process Graphic composites and displays. The user builds graphic displays and selects elements and attributes that are to be given dynamic behavior.

The Smart Process Graphics Editor also enables developers to create specialized interfaces that are sophisticated enough to contain dynamic behaviors and data driven elements that are not sourced from traditional DeltaV data sources. For example, the Device, Batch, and Advanced Control teams will be able to build application views for each of their applications and tie those views to Web Services provided by each of the respective teams.

1.3.1.2 Template Builder

Template Builder lets graphic configuration folks manipulate the complete set of dynamic data bindings contained inside a single process graphic display. From this view configuration folks can specify the nature of the required data and assign parameters and logic to dynamic objects in the graphic displays.

1.3.1.3 Data Mapper

Data Mapper enables application developers to define connections between the inputs of an XML template and data by doing the following tasks:

- Selecting the XML elements to map
- Specifying the sample XML data file to which the XML template will be mapped
- Connecting the inputs of the XML template to the XML data file

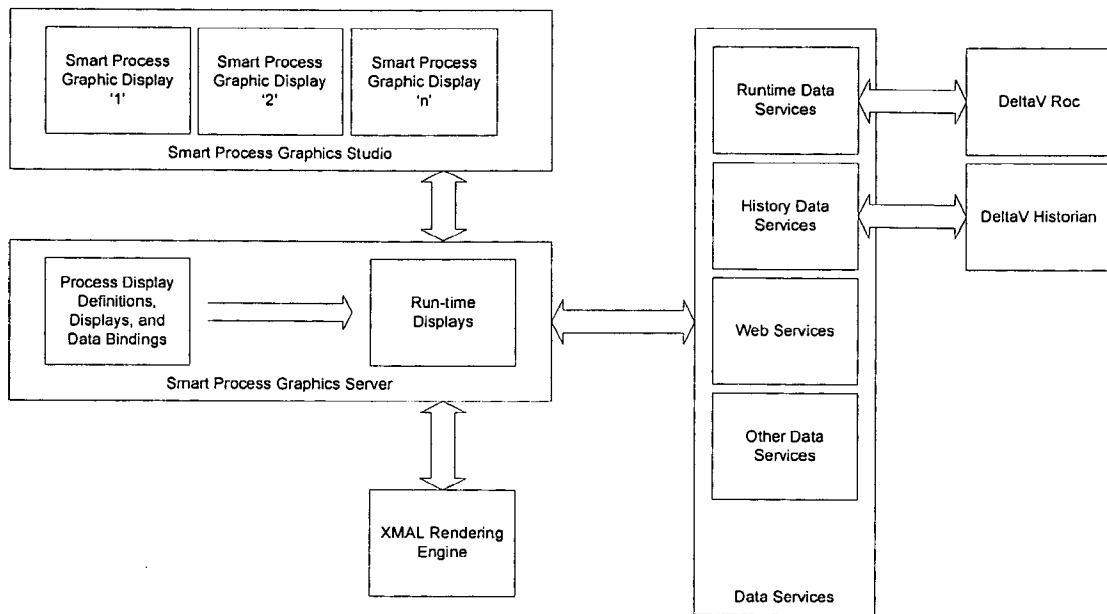
Multiple data sources including DeltaV runtime, DeltaV Historical, XML files, ODBC, Web services, etc can be defined. The data from these sources can be linked with process graphics.

1.3.2 Process Graphics Server

The Process Graphics Server is designed to be a high-performance platform for the management and transformation of application data and images. It retrieves information from multiple data sources and serves the data to the Process Graphic Displays.

1.4 In this concept document

The Smart Process Graphics Architecture includes both Configuration and Runtime applications. These applications are identified above as Process Graphics Studio and Process Graphics Server. These applications are supported by configuration and runtime subsystems and components. This is illustrated in the drawing below.



This document focuses on the configuration aspects of Smart Process Graphics. The runtime aspects will be described in a separate document.

1.5 The Process Graphics Editor

1.5.1

1.6 The Process Graphics Runtime

1.6.1

1.7 References

- 1- Process Graphics Editor
- 2- IFIX and FIX32
- 3- Corel Smart Graphics Studio
- 4- Mathematica 5
- 5- Longhorn
- 6- Customer Feedback

2 Process Graphics Display Editor Requirements

2.1 General Requirements

2.2 Graphic Primitives

2.2.1 UI Controls

Buttons
Checkbox

2.2.2 Behaviors

Actions
Alerts

2.2.3 Attributes

Context Menu
Drag and Drop

2.2.4 Elements

Circle
Rectangle

2.2.5 Commands

Cut
Copy
Paste
Move Forward / Backward

2.3 *Parameter References*

Internal
External
Aliased

2.4 Binding to data sources

2.4.1 Create Input / Output

2.4.2 Define Functionality of Dynamic Elements

2.4.3 Binding Input / Outputs to Functions

2.5 Projects

2.6 Displays

2.7 Groups

2.8 Composites

2.9 Faceplates

2.10Dynamos

2.11Data Binding Sources

2.11.1 DeltaV Runtime

2.11.2 DeltaV Historian

2.11.3 DeltaV Alarms

2.11.4 DeltaV Events

2.11.5 OPC

2.11.6 XML File

2.11.7 HTTP Browser Link

2.11.8 General Web Services

2.11.9 Yukon Managed Runtime via ADO

2.11.10 ODBC

2.12 Scripting

2.13 Display Format

2.13.1 XMAL format

2.13.2 Canvas Size

2.13.3 References

2.14 Controls

2.14.1 Real-time Trends

2.14.2 Historical Plots

2.14.3 Charts

2.14.4 Alarm Banner

2.14.5 Alarm Summary

2.14.6 Advanced Control

2.14.7 Batch

2.15 Tag Group Editor

2.16 Multimedia

2.16.1 Sound

2.16.2 Video

2.17 *System Preferences*

2.17.1 Color

2.17.2 Fonts

2.17.3 Locale

2.18 *Reports*

2.19 *Diagnostics*

2.20 *Performance Checking Displays*

2.21 *Experts*

3 Process Graphics Display Requirements

3.1 *General Requirements*

4 Platform Requirements

4.1 *Tablet*

4.2 *Multi-screen Configurations*

4.3 *Handhelds*

5 Process Graphics Runtime Requirements

Requirements that affect the PGE.

5.1 *Loading a Display / Project at Runtime*

5.2 *Preloading Displays*

5.3 *Display Navigation*

5.4 *Custom Keyboard*

5.5 *Multi-screen Monitor*

6 Scenarios

6.1 User Creates Display

6.2 Building a Display

6.3 Building a Composite

6.4 Working with Controls

7 Process Graphic Design

7.1 Configuration Workspace

7.1.1 Multi-document Interface

7.1.2 Projects

7.2 Displays

7.2.1 Format

7.2.2 Linking

7.3 Menus and Toolbars

7.3.1 Menus

Open
Save
Close
Copy

7.3.2 Toolbars

7.4 Views

7.5 Controls

7.6 Interface with Configuration

7.7 Interface with Runtime

7.8 Display Quickview

7.9 Project Files

XMAL File Format
References
Bitmaps
Other project files

7.10

Process Graphics

Edit, Save, Download, Run

General Scenario

Table of Contents:

1	INTRODUCTION.....	194
1.1	USE CASES.....	194
1.2	OVERVIEW OF THE PROCESS GRAPHICS SYSTEM.....	194
1.2.1	General Overview	194
1.2.2	Configuration and Runtime Constructs.....	194
1.2.3	Models.....	196
1.2.4	Configuration Model.....	196
1.2.5	Runtime Model.....	197
1.2.6	Configuration Workspace / Tools.....	198
1.2.7	Summary of Files	199
1.3	BACKGROUND FOR SCENARIOS	200
1.3.1	Sample Configuration Used in the Scenarios	200
2	SCENARIO – STATIC AND DYNAMIC TEXT	201
2.1	CREATE DISPLAY 'DISP_1'.....	201
2.1.1	Preconditions	201
2.1.2	Postconditions.....	201
2.1.3	Scenario	201
2.1.4	Component Sequence Diagram	202
2.2	DOWNLOAD DISPLAY 'DISP_1'	202
2.2.1	Preconditions	202
2.2.2	Postconditions.....	202
2.2.3	Scenario	202
2.2.4	Component Sequence Diagram	203
2.3	LOAD DISPLAY 'DISP_1' INTO RUNTIME	203
2.3.1	Preconditions	203
2.3.2	Postconditions.....	203
2.3.3	Scenario	203
2.3.4	Component Sequence Diagram	204
2.4	PROCESS UPDATES DISPLAY 'DISP_1'	204
2.4.1	Preconditions	204
2.4.2	Postconditions.....	204
2.4.3	Scenario	204
2.4.4	Component Sequence Diagram	204
3	SCENARIO - DATA ENTRY FIELD.....	205
3.1	CREATE DATA ENTRY FIELD.....	205
3.1.1	Preconditions	205
3.1.2	Postconditions.....	205
3.1.3	Scenario	205
3.1.4	Component Sequence Diagram	205
3.2	DOWNLOAD DATA ENTRY FIELD	205
3.2.1	Preconditions	205
3.2.2	Postconditions.....	205
3.2.3	Scenario	205
3.2.4	Component Sequence Diagram	206
3.3	LOAD DATA ENTRY FIELDS INTO RUNTIME	206
3.3.1	Preconditions	206
3.3.2	Postconditions.....	206
3.3.3	Scenario	206
3.3.4	Component Sequence Diagram	206
3.4	USER CHANGES VALUE THROUGH DATA ENTRY FORM.....	207
3.4.1	Preconditions	207
3.4.2	Postconditions.....	207
3.4.3	Scenario	207

3.4.4	Component Sequence Diagram	207
4	SCENARIO - GRAPHIC ITEM PUMP.....	207
4.1	CREATE GRAPHIC ITEM 'PUMP_1'	207
4.1.1	Preconditions	207
4.1.2	Postconditions.....	207
4.1.3	Scenario	207
4.1.4	Component Sequence Diagram	208
4.2	DOWNLOAD DISPLAY WITH GRAPHIC ITEM 'PUMP1'	208
4.2.1	Preconditions	208
4.2.2	Postconditions.....	208
4.2.3	Scenario	208
4.2.4	Component Sequence Diagram	208

1 Introduction

This document describes a few general 'Edit-Save-Download-Run' Process Graphics Session.

1.1 Use Cases

The Use Cases that are described in this document include:

1. Using Static and Dynamic Text
2. Using Data Entry Fields, in this case a simple Text Entry Box
3. Using Graphic Elements, in this case a Pump

1.2 Overview of the Process Graphics System

1.2.1 General Overview

The overall Process Graphics Architecture is made up of a combination of Configuration and Runtime Components. These components are supported by other subsystems such Control, Communications, etc. The Configuration System is linked to the runtime through a supporting Runtime workspace and a set of pre-built files providing a set of known behaviors. This set of relationships is illustrated below.

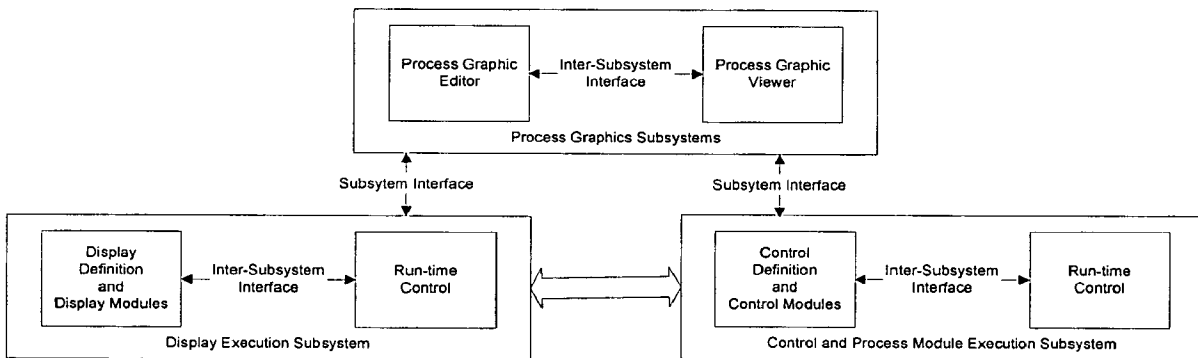


Figure 1. Process Module Class "Sub-system Relationships"

1.2.2 Configuration and Runtime Constructs

Similar to Control Strategy configuration there are Primitives, Composites, Classes, and Templates. These are described here.

Primitives – pre-built components that are built and distributed as components. Primitives have built-in behavior. In some cases the behavior is very simple, add two things together, while in other cases the behavior is quite complex as in a BOI control. Each primitive in the editor has an SVG equivalent (although one editor primitive may be translated into multiple SVG objects, e.g. multi-line text, gradient filled objects etc.).

Library Composites - pre-built collections of primitives and scripting. We will provide libraries of composites. Users can also create their own library of composites. These library composites can be composed of any arbitrary collection of display fragments, including static and dynamic components.

Classes - Component classes correspond to reusable display composites. When an instance of a class is created on another display the user can override the public parameters defined in the class, together with its viewport characteristics (position on screen, size).

Templates - pre-built collections of primitives and scripting. Templates are libraries of pre-built displays in the library which can be used as the starting point for creating new displays.

The arrangement of constructs is illustrated in the illustration below:

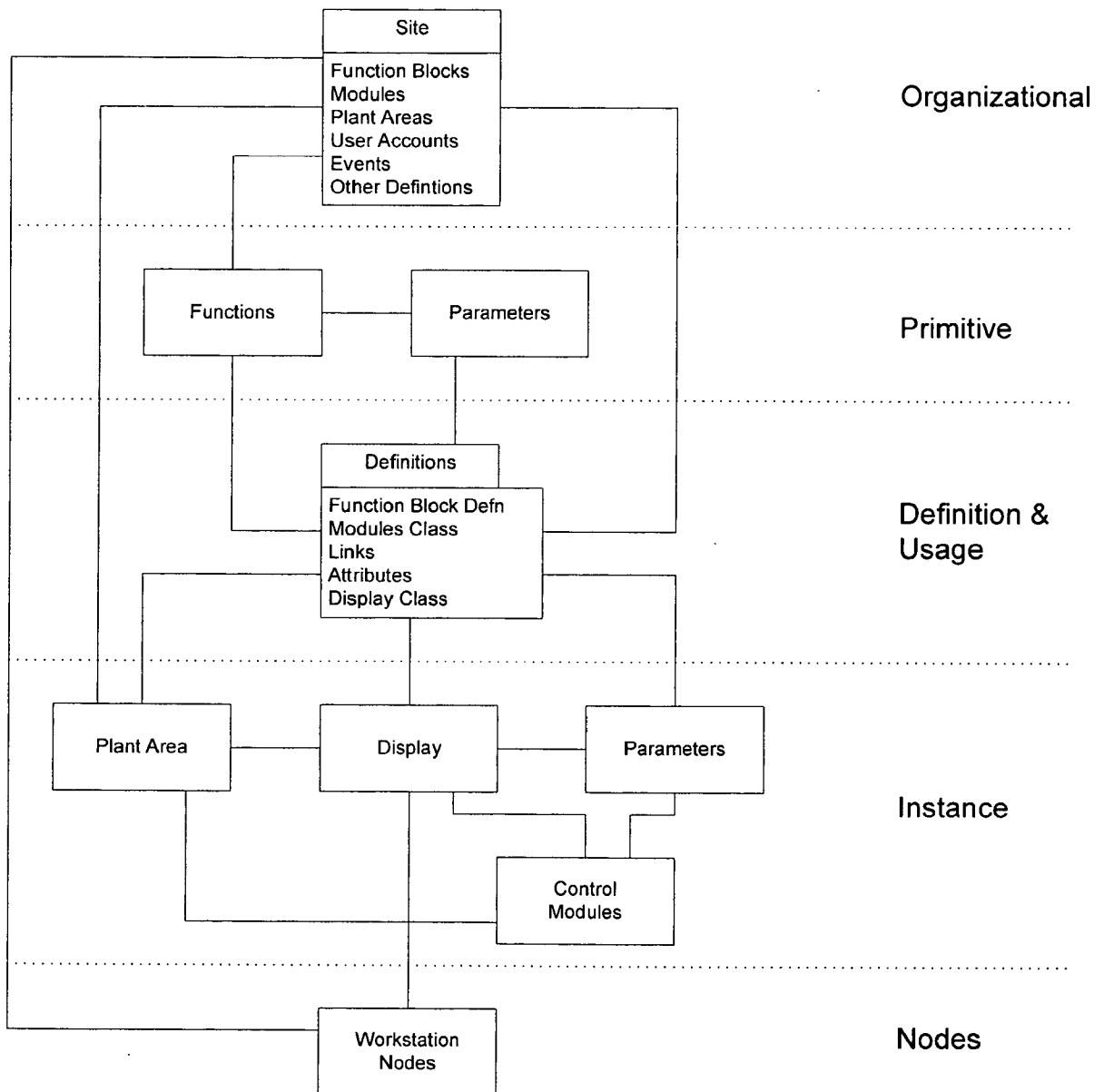


Figure 2. Display Model

1.2.3 Models

1.2.4 Configuration Model

The scenarios in this document make reference to the following model:

Figure 3. Configuration Model

1.2.5 Runtime Model

The scenarios in this document make reference to the following model:

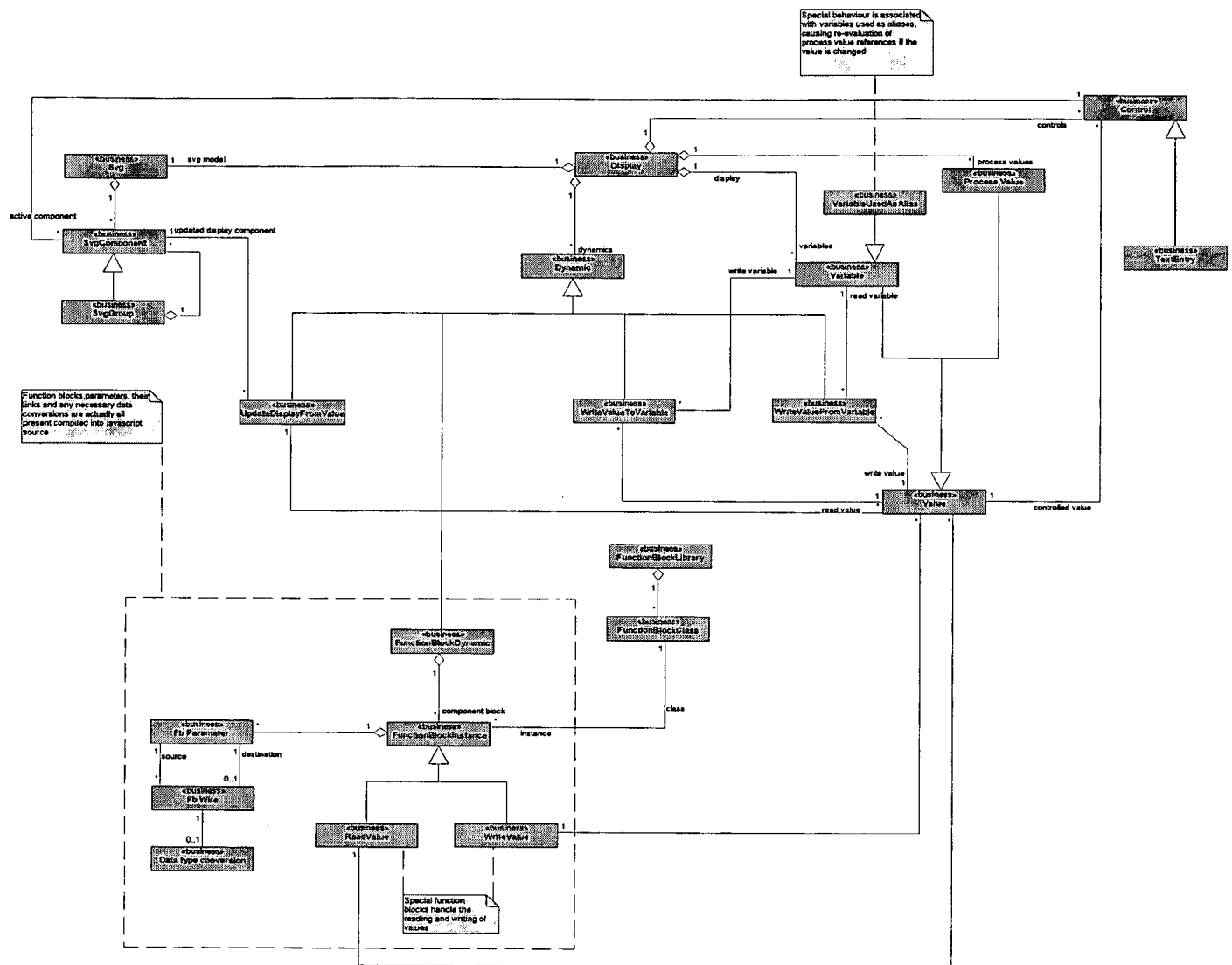


Figure 4. Runtime Model

1.2.6 Configuration Workspace / Tools

The configuration workspace used in these scenarios makes use of a number of features. A snapshot of the editor is included below.

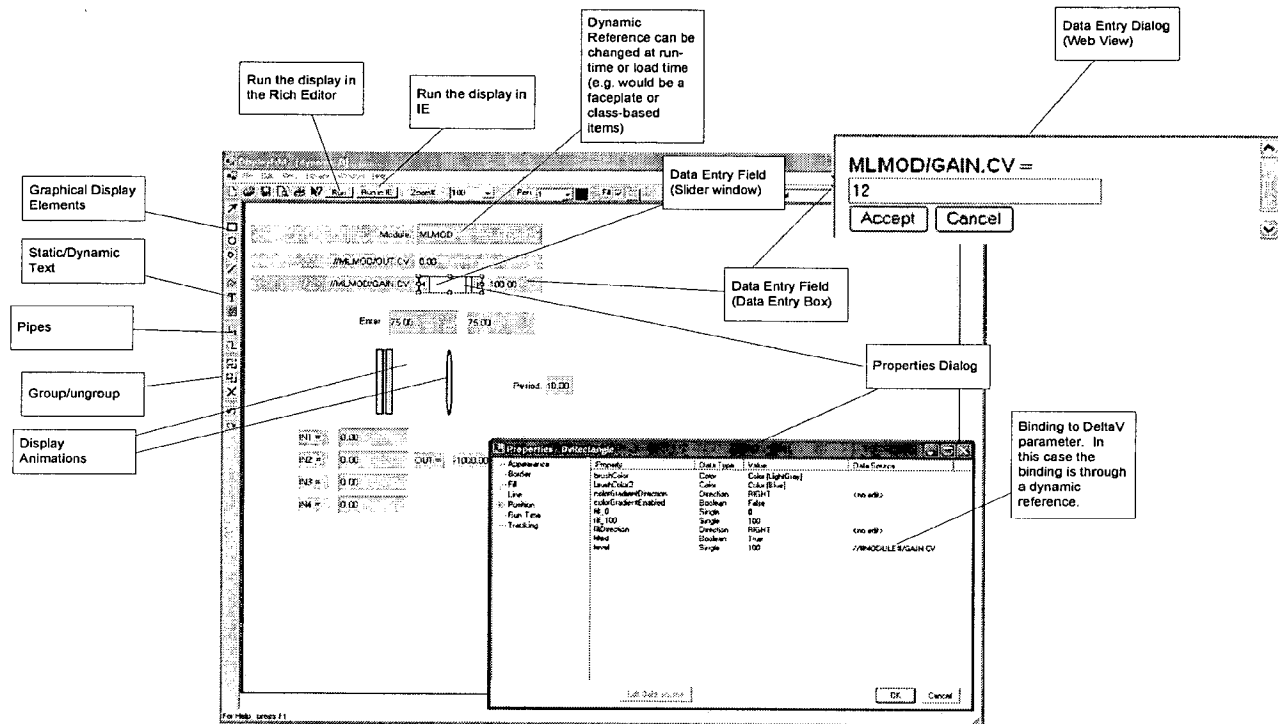


Figure 5. Process Graphics Configuration Tools

1.2.7 Summary of Files

There are a number of files used in the Process Graphics scenarios discussed in this document. These files support both Rich (Win Forms) and Thin Clients (IE, Handhelds). Rich Clients can directly load the native XML format. Thin Clients make use of a combination of SVG and Java scripting. Using display 'dynamic' the following files are used:

dynamic.SVG/XMAL	The Rich Client files will be in XML format.
OpenDisplay.aspx	Initial HTML file – used to select display to be displayed.
OpenDisplay.js	Works with OpenDisplay.aspx to open initial display.
FunctionBlocks.js	Provides display element dynamic behavior.
dynamic.dvg.SVG/XMAL	SVG/XMAL format of dynamic.SVG/XMAL file.
dynamic.dvg.js	Provides dynamic [i.e. scripting] behavior for display 'dynamic.SVG/XMAL'

The relationship of the files is illustrated and discussed below.

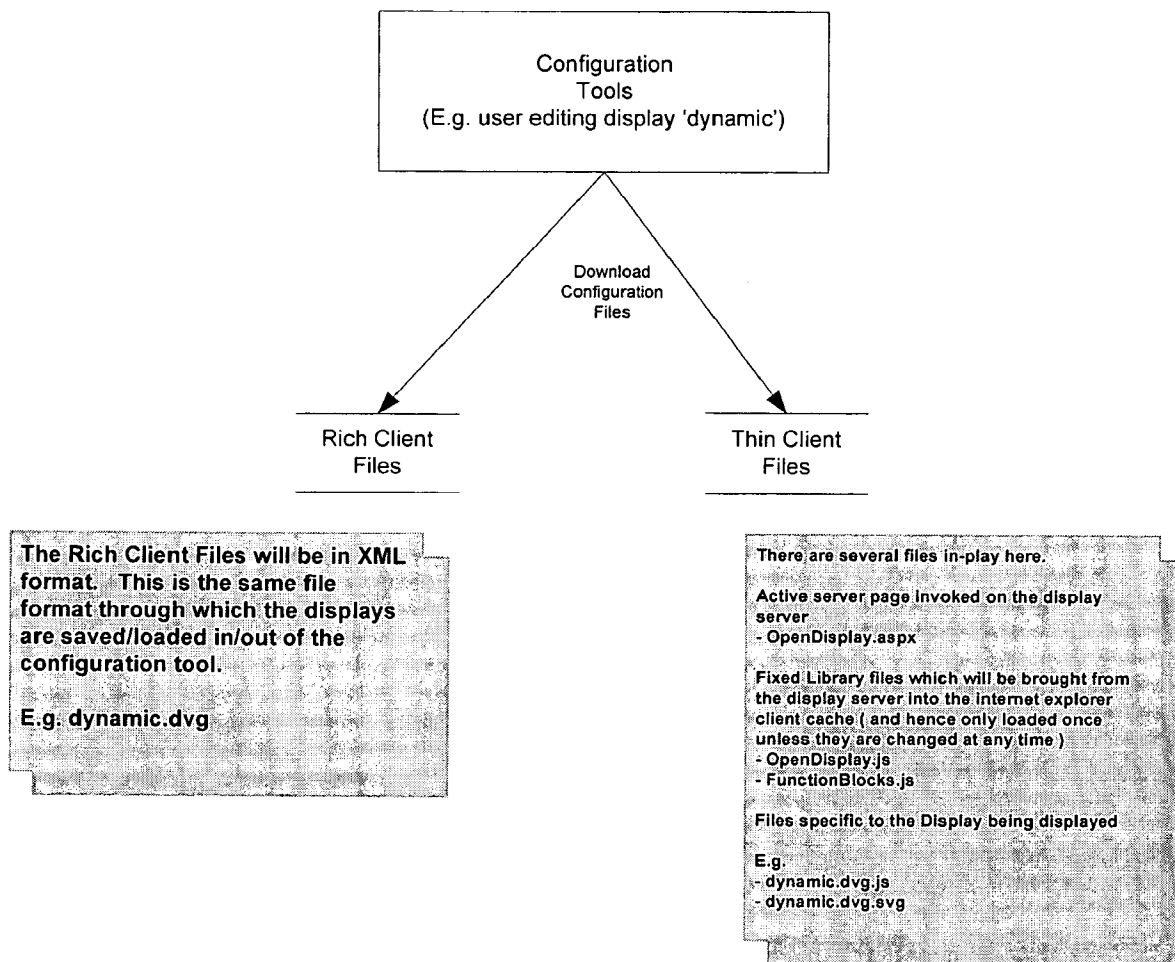


Figure 6. Display Storage

1.3 Background for Scenarios

1.3.1 Sample Configuration Used in the Scenarios

Site

+ Library

+ System Configuration

+ Recipes

+ Setup

+ Control Strategies

+ AREA_A

+ Display Directory

+ DISP_1

+ MLMOD

+ Physical Network

+ Control Network

+ USAUST-DELLBERT

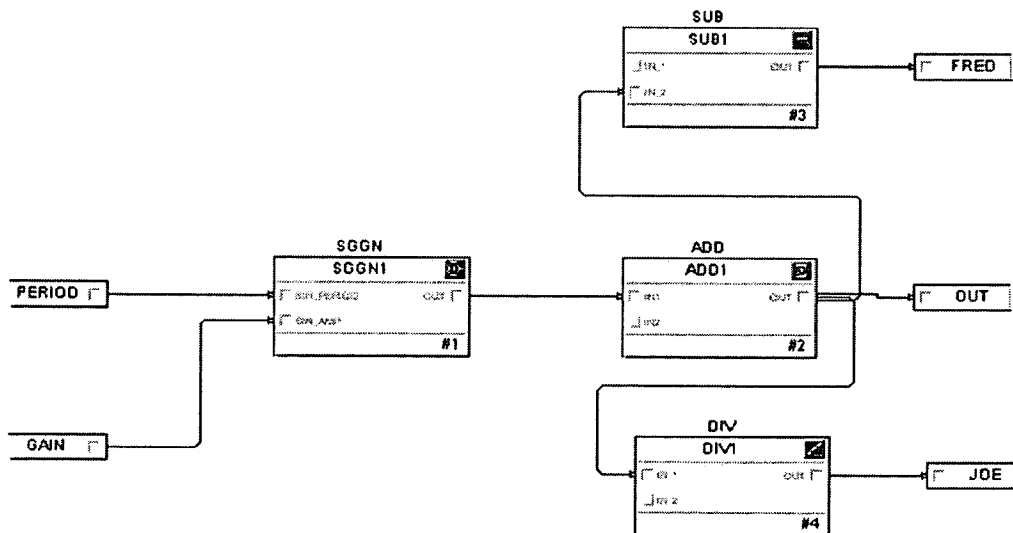
+ Assigned Modules

+ MLMOD

+ Operator

+ OVERVIEW

The Control Module 'MLMOD' contains the following algorithm:



2 Scenario – Static and Dynamic Text

2.1 Create Display 'DISP_1'

2.1.1 Preconditions

- 1- Explorer View is Started
- 2- Area 'AREA_A' exists
- 3- Module 'MLMOD' exists

2.1.2 Postconditions

- 1- Display 'DISP_1' created with Static and Dynamic Text Fields added to display
- 2- Display 'DISP_1' is executing in the runtime
- 3- Display 'DISP_1' is receiving and processing updates

2.1.3 Scenario

- 1- User creates Display 'DISP_1' under Display Directory of Area 'AREA_A'
- 2- User creates static text item on Process Graphic 'DISP_1'.
 - a. User selects 'Text Item' on pallet and draws a Text Box. A text object is created at the cursor and stretched until the user releases mouse.

NOTE: The text object is added to the display model at the active display root. This display root is the root for the current drill down level. If not drilled down this is the root of the display.
 - b. User fills the static text box with the text 'LOOP-101'.
- 3- User creates dynamic text item on Process Graphic 'DISP_1'. The value is to be directly bound to process value '//MLMOD/OUT.CV'
 - a. User selects 'Text Item' on pallet and draws a Text Box. A text object is created at the cursor and stretched until the user releases mouse.
 - b. The user selects the dynamics menu context menu option of an existing text object.
 - c. The user selects "Add"
 - d. The user picks the "textFloat" property, and sets the data source to "//MODULE/ATTR.FIELD", in this case "//MLMOD/OUT.CV".
 - e. The user optionally sets offset and scaling.
 - f. The user OKs the dialogs
- 4- User Saves Display 'DISP_1' in Area 'AREA_A' (note, displays can be saved into a particular context, e.g. Area, Unit, Library). The display is stored into the database as an XML document. The top level public parameters of the display are extracted and represented as DB objects which can be manipulated through the explorer. The status of the display is initially set to "Not published".

2.1.4 Component Sequence Diagram

2.2 Download Display 'DISP_1'

2.2.1 Preconditions

- 1- Display 'DISP_1' has been created
- 2- Display 'DISP_1' has not been assigned or downloaded

2.2.2 Postconditions

- 1- Display 'DISP_1' is assigned to node 'USAUST-DELLBERT'
- 2- Display 'DISP_1' has been downloaded

2.2.3 Scenario

- 1- User Assigns Display 'DISP_1' to Workstation 'USAUST-DELLBERT'.
- 2- The user changes the status of the display to published. This causes the SVG and JS script to be generated. Since these scripts have never been downloaded (in fact if these scripts differed from the previous scripts), the display is marked as requiring download.
 - a. SVG script 'DISP_1.dvg.svg' is created

```
<?xml version="1.0" encoding="utf-8" standalone="yes"?>
<svg xmlns:dv="abcdefg" id="id_SVG" width="1024" height="768" onload="loadSvg(evt);">
  <defs />
  <g dv:type="canvas" dv:update-period="500">
    <rect width="1024" height="768" fill="#ebebeb" />
    <g dv:type="none">
      <rect x="2" y="2" width="1020" height="764" fill="none" stroke="#000000" stroke-width="1" />
    </g>
  </g>
  <g>
    <g>
      <rect x="50" y="50" width="70" height="25" fill="#d3d3d3" />
      <text x="50" y="66" fill="#000000" text-anchor="start">LOOP-101</text>
    </g>
    <rect x="130" y="50" width="70" height="25" fill="#d3d3d3" />
    <text x="130" y="74" fill="#000000" text-anchor="start">0.00
      <dv:dynamics id="id_DYNAMIC0" tag="MLMOD/OUT.CV" attribute="text"
valueConditioning="val=fmt(val,2)" />
    </text>
  </g>
</g>
</svg>
```

- a.
 - b. Java script 'DISP_1.dvg.js' is created

```
updatePeriodMs=500;
```

- 3- User selects downloads display
 - a. The download process transfers the SVG and JS scripts to the display web server.
 - i. SVG script 'DISP_1.dvg.svg' is transferred to the display server (i.e. Starburn Server & PRO+ nodes).

- ii. Java script 'DISP_1.dvg.js' is transferred to the display server (i.e. Starburn Server & PRO+ nodes).
- b. A download process transfers list holding set of displays and over-rides to their parameters is generated
 - i. A download script giving a list of displays for workstation 'USAUST-DELLBERT' is downloaded to 'USAUST-DELLBERT'
 - ii. A download script providing list of over-rides for display 'DISP_1' is downloaded to node 'USAUST-DELLBERT'

2.2.4 Component Sequence Diagram

2.3 Load Display 'DISP_1' into Runtime

2.3.1 Preconditions

- 1- Display 'DISP_1' has been downloaded to workstation 'USAUST-DELLBERT'
- 2- Module 'MLMOD' has been downloaded

2.3.2 Postconditions

- 1- Display 'DISP_1' loaded
- 2- Dynamic Text fields are updating

2.3.3 Scenario

- 1- User loads the display into the runtime
 - a. The display web server receives a URL to openDisplay.aspx specifying the display to open and any parameter overrides.
 - b. openDisplay.aspx constructs a web page built up of
 - i. Javascript to provide dynamic behavior
 - 1. OpenDisplay.js - the JavaScript library of methods to parse and control a display and its communications
 - 2. FunctionBlocks.js - the library of function blocks available for scripting behavior Parameter overrides for the display
 - 3. DISP_1.dvg.js – the display specific JavaScript
 - ii. Embedded SVG viewer
 - 1. The display specific SVG file 'DISP_1.dvg.svg'. The SVG file contains all the SVG content required to statically render the display
 - 2. a definition of all the dynamic behaviour using DeltaV specific tags
- 2- Static parts are loaded - the Embedded SVG viewer will render the initial view from the SVG definition.
- 3- Dynamic links are identified and bound –

- a. The onload processing will then fire off the javascript which searches the SVG for deltaV tags which identify dynamic behavior. Depending upon the type of dynamic (directly bound, user updatable, fb algorithm etc) structures are built up to allow rapid traversal of the dynamic components.
- 4- There are components for registering for updates from the runtime
 - a. All aliases are resolved, the process value paths are extracted , and the list is sent to the data server to register for updates.
 - b. A session ID is returned to the JS code in the web page. If at any time the value of any of the aliases in a display is changed (by the user or by an algorithm) then the current update registration is cancelled and a new list of process value paths is extracted and registered for updates.

2.3.4 Component Sequence Diagram

2.4 Process Updates Display 'DISP_1'

2.4.1 Preconditions

- 1- Display 'DISP_1' has been loaded
- 2- Display 'DISP_1' has registered for updates

2.4.2 Postconditions

- 1- Display 'DISP_1' processed update from the field
- 2- Dynamic Text Field updated

2.4.3 Scenario

- 1- An update to '//MLMOD/OUT.CV' is received from the runtime.
- 2- The update is received and stored as a new value for the parameter.
- 3- At the display refresh period, all defined dynamic behaviors within the display are executed.
- 4- Dynamic behaviors update variables which in turn result in SVG properties being updated.
- 5- User sees updates to the display.
- 6- If any of the processing results in changes to process values, then the process values are sent to the data server which then pushes them through the OPC interface into the runtime. The new value is reflected back to the web page through the usual periodic update mechanism. If any of the aliases are changed then update registrations are updated as described above.

2.4.4 Component Sequence Diagram

3 Scenario - Data Entry Field

3.1 Create Data Entry Field

3.1.1 Preconditions

- 1- Explorer View is Started
- 2- Area 'AREA_A' exists
- 3- Module 'MLMOD' exists

3.1.2 Postconditions

- 1- Display 'DISP_1' created
- 2- Static and Dynamic Text Fields added to display

3.1.3 Scenario

- 1- User ...
- 2- The following is created...

```
<?xml version="1.0" encoding="UTF-8"?>
< Results schema="1.0">
  <Site name="The Site">
    <PlantAreas>
      </PlantAreas>
    </Site>
  </Results>
```

- 3- Explorer View Shows Process Cell 'LINE_1'

3.1.4 Component Sequence Diagram

3.2 Download Data Entry Field

3.2.1 Preconditions

- 1- Explorer View is Started
- 2- Area 'AREA_A' exists
- 3- Module 'MLMOD' exists

3.2.2 Postconditions

- 1- Display 'DISP_1' created
- 2- Static and Dynamic Text Fields added to display

3.2.3 Scenario

- 1- User ...

- 2- The following is created...

```
<?xml version="1.0" encoding="UTF-8"?>
< Results schema="1.0">
  <Site name="The Site">
    <PlantAreas>
    </PlantAreas>
  </Site>
</Results>
```

- 3- Explorer View Shows Process Cell 'LINE_1'

3.2.4 Component Sequence Diagram

3.3 Load Data Entry Fields into Runtime

3.3.1 Preconditions

- 1- Explorer View is Started
- 2- Area 'AREA_A' exists
- 3- Module 'MLMOD' exists

3.3.2 Postconditions

- 1- Display 'DISP_1' created
- 2- Static and Dynamic Text Fields added to display

3.3.3 Scenario

- 1- User ...
- 2- The following is created...

```
<?xml version="1.0" encoding="UTF-8"?>
< Results schema="1.0">
  <Site name="The Site">
    <PlantAreas>
    </PlantAreas>
  </Site>
</Results>
```

- 3- Explorer View Shows Process Cell 'LINE_1'

3.3.4 Component Sequence Diagram

3.4 User Changes Value Through Data Entry Form

3.4.1 Preconditions

- 4- Explorer View is Started
- 5- Area 'AREA_A' exists
- 6- Module 'MLMOD' exists

3.4.2 Postconditions

- 3- Display 'DISP_1' created
- 4- Static and Dynamic Text Fields added to display

3.4.3 Scenario

- 4- User ...
- 5- The following is created...

```
<?xml version="1.0" encoding="UTF-8"?>
< Results schema="1.0">
  <Site name="The Site">
    <PlantAreas>
    </PlantAreas>
  </Site>
</Results>
```

- 6- Explorer View Shows Process Cell 'LINE_1'

3.4.4 Component Sequence Diagram

4 Scenario - Graphic Item Pump

4.1 Create Graphic Item 'PUMP_1'

4.1.1 Preconditions

- 1- Explorer View is Started
- 2- Area 'AREA_A' exists
- 3- Module 'MLMOD' exists

4.1.2 Postconditions

- 1- Display 'DISP_1' created
- 2- Static and Dynamic Text Fields added to display

4.1.3 Scenario

- 1- User ...

- 2- The following is created...

```
<?xml version="1.0" encoding="UTF-8"?>
< Results schema="1.0">
  <Site name="The Site">
    <PlantAreas>
    </PlantAreas>
  </Site>
</Results>
```

- 3- Explorer View Shows Process Cell 'LINE_1'

4.1.4 Component Sequence Diagram

4.2 Download Display With Graphic Item 'PUMP1'

4.2.1 Preconditions

- 1- Explorer View is Started
- 2- Area 'AREA_A' exists
- 3- Module 'MLMOD' exists

4.2.2 Postconditions

- 1- Display 'DISP_1' created
- 2- Static and Dynamic Text Fields added to display

4.2.3 Scenario

- 1- User ...
- 2- The following is created...

```
<?xml version="1.0" encoding="UTF-8"?>
< Results schema="1.0">
  <Site name="The Site">
    <PlantAreas>
    </PlantAreas>
  </Site>
</Results>
```

- 3- Explorer View Shows Process Cell 'LINE_1'

4.2.4 Component Sequence Diagram

Load Display With Graphic Item 'PUMP1'

Preconditions

Explorer View is Started

Area 'AREA_A' exists
Module 'MLMOD' exists
Postconditions
Display 'DISP_1' created
Static and Dynamic Text Fields added to display
Scenario
User ...
The following is created...

```
<?xml version="1.0" encoding="UTF-8"?>  
< Results schema="1.0">  
  <Site name="The Site">  
    <PlantAreas>  
    </PlantAreas>  
  </Site>  
</Results>
```

Explorer View Shows Process Cell 'LINE_1'

Component Sequence Diagram

Process Updates for Graphic Item 'PUMP_1'
Preconditions
Explorer View is Started
Area 'AREA_A' exists
Module 'MLMOD' exists
Postconditions
Display 'DISP_1' created
Static and Dynamic Text Fields added to display
Scenario
User ...
The following is created...

```
<?xml version="1.0" encoding="UTF-8"?>  
< Results schema="1.0">  
  <Site name="The Site">  
    <PlantAreas>  
    </PlantAreas>
```

```
</Site>
</Results>
```

Explorer View Shows Process Cell 'LINE_1'

Component Sequence Diagram

User Sends Command 'Start' to Graphic Item 'PUMP1'

Preconditions

Explorer View is Started

Area 'AREA_A' exists

Module 'MLMOD' exists

Postconditions

Display 'DISP_1' created

Static and Dynamic Text Fields added to display

Scenario

User ...

The following is created...

```
<?xml version="1.0" encoding="UTF-8"?>
< Results schema="1.0">
  <Site name="The Site">
    <PlantAreas>
    </PlantAreas>
  </Site>
</Results>
```

Explorer View Shows Process Cell 'LINE_1'

Component Sequence Diagram

Configuration Query and Command Engine

Table of Contents:

1	INTRODUCTION.....	214
1.1	QUERY AND COMMAND OBJECTIVES.....	214
1.2	USE CASES.....	214
2	GENERAL CONFIGURATION SCENARIOS.....	215
2.1	BACKGROUND FOR SCENARIOS.....	215
2.1.1	Sample Hierarchy.....	215
2.1.2	Component Hierarchy.....	216
2.2	EXPAND AREA 'AREA_A'.....	217
2.2.1	Preconditions.....	217
2.2.2	Postconditions.....	217
2.2.3	Scenario.....	217
2.2.4	Component Sequence Diagram.....	218
2.3	EXPAND PROCESS CELL 'LINE_1'.....	221
2.3.1	Preconditions.....	221
2.3.2	Postconditions.....	221
2.3.3	Scenario.....	221
2.3.4	Component Sequence Diagram.....	221
2.4	CREATE PROCESS CELL 'LINE_2' UNDER AREA 'AREA_A'.....	222
2.4.1	Preconditions.....	222
2.4.2	Postconditions.....	222
2.4.3	Scenario.....	222
2.4.4	Component Sequence Diagram.....	223
2.5	CREATE DI CARD UNDER CONTROLLER 'CTRL1'.....	223
2.5.1	Preconditions.....	223
2.5.2	Postconditions.....	223
2.5.3	Scenario.....	223
2.5.4	Component Sequence Diagram.....	224
2.6	CONVERT FB 'AI1' FROM DEFN 'AI' TO 'FFAI' (W/O UPDATES).....	224
2.6.1	Preconditions.....	224
2.6.2	Postconditions.....	224
2.6.3	Scenario.....	225
2.6.4	Component Sequence Diagram.....	225
2.7	CONVERT FB 'AI1' FROM DEFN 'AI' TO 'FFAI' (W/ UPDATES).....	225
2.7.1	Preconditions.....	226
2.7.2	Postconditions.....	226
2.7.3	Scenario.....	226
2.7.4	Component Sequence Diagram.....	228
2.8	CHANGE PROCESS CELL NAME FROM 'LINE_1' TO 'UTIL' (W/ UPDATES).....	228
2.8.1	Preconditions.....	228
2.8.2	Postconditions.....	228
2.8.3	Scenario.....	228
2.8.4	Component Sequence Diagram.....	229
2.9	CHANGE NAME OF FBU 'AI1' TO 'FLOW1' (W/UPDATES).....	230
2.9.1	Preconditions.....	230
2.9.2	Postconditions.....	230
2.9.3	Scenario.....	230
2.9.4	Component Sequence Diagram.....	232
2.10	FIND ALL MODULES WITH PARAMETER 'MSTATUS_MASK' WITH BIT 4 SET IN PLANT AREA 'AREA_A'.....	232
2.10.1	Preconditions.....	232
2.10.2	Postconditions.....	232
2.10.3	Scenario.....	232
2.10.4	Component Sequence Diagram.....	233

2.11	FIND ALL MODULES OF TYPE 'PLM' IN PLANT AREA 'AREA_A'	233
2.11.1	Preconditions	233
2.11.2	Postconditions	233
2.11.3	Scenario	233
2.11.4	Component Sequence Diagram	234
3	TECHNICAL ARCHITECTURE	235
3.1	OVERVIEW	235
3.2	COMPONENT ARCHITECTURE	235
3.3	SUBSYSTEMS	236
4	CONFIGURATION APPLICATIONS AND SUPPORTING FRAMEWORKS.....	237
4.1	OVERVIEW	237
4.2	CLIENT COMPONENT FRAMEWORK	239
4.2.1	Object Model	239
4.2.2	Properties.....	239
4.2.3	Children.....	239
4.2.4	Commands.....	240
4.3	CLIENT/SERVER	242
4.3.1	Connectivity	242
4.3.2	Threading	243
4.4	QUERY FRAMEWORK	245
4.4.1	Description	245
4.4.2	Updating Queries	246
4.5	COMMAND FRAMEWORK	246
4.6	SERVICE FRAMEWORK	246
4.6.1	Import	246
4.7	SERVER CONFIGURATION MODEL.....	247
4.8	UPDATE NOTIFICATION	248
4.8.1	Object Model.....	248
4.8.2	Register a Query.....	248
4.8.3	Notify Thread	249
4.8.4	Execute a Command	250
4.8.5	Deregister a Query.....	251

1 Introduction

1.1 Query and Command Objectives

1. Improve Overall Configuration Integration
 - One framework for all tools
 - Common controls across all tools
 - Large System configuration performance improvements
2. Support Multiple Development Centers

1.2 Use Cases

1. Read
 - Find
 - Verify Module
 - Bulk Read
 - Load Graphic
2. Command
 - Create Card
 - Rename Module
 - Delete – Block Usage, Process Cell, Parameter
 - Set Properties
 - Enable Device Alerts
 - Connect Block to Fieldbus
 - Instantiate a Block
 - New Card
 - Save Module
 - Save Graphic
3. Services
 - Import/Export
 - Download
 - Find
 - Verify
 - Bulk Write/Update

2 General Configuration Scenarios

2.1 Background for Scenarios

2.1.1 Sample Hierarchy

Several of the scenarios make use of the following hierarchy:

Site

+ Library

+ System Configuration

+ Recipes

+ Setup

+ Control Strategies

+ AREA_A

+ LINE_1

+ BOILER_A

+ WATER_INLET_FLOW

+ VALVE_1

+ FLOW_METER_1

AI1 -> FFAI1

+ PLM_1

+ Physical Network

+ Control Network

+ CTRL1

+ I/O

+ C01

+ CH01

+ CH02

:

+ C01

+ P01

+ PDT1

FFAI1

FFAI2

2.1.2 Component Hierarchy

The scenarios in the following sections make use of several new components and interfaces. These components provide Query Services for clients. The Query and Command Components are the topic of the following scenarios. The specialized services are not covered in this document.

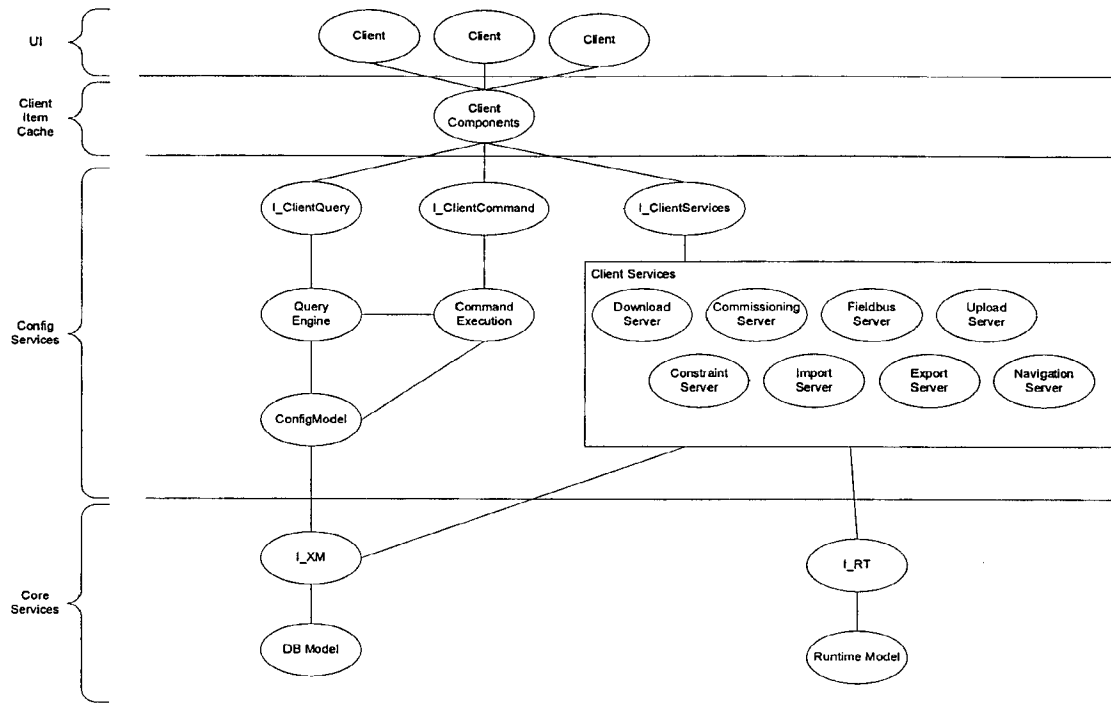


Figure 1. Component Layers

2.2 Expand Area 'AREA_A'

2.2.1 Preconditions

- 1- Explorer View is Started
- 2- Area 'AREA_A' not expanded.

2.2.2 Postconditions

- 1- Area 'AREA_A' expanded
- 2- Explorer shows Process Cell 'LINE_1'

2.2.3 Scenario

- 1- User selects Area 'AREA_A'
- 2- App generates query

Site.PlantAreas[name='AREA_A']().SubEquipmentItems(name,type,desc)

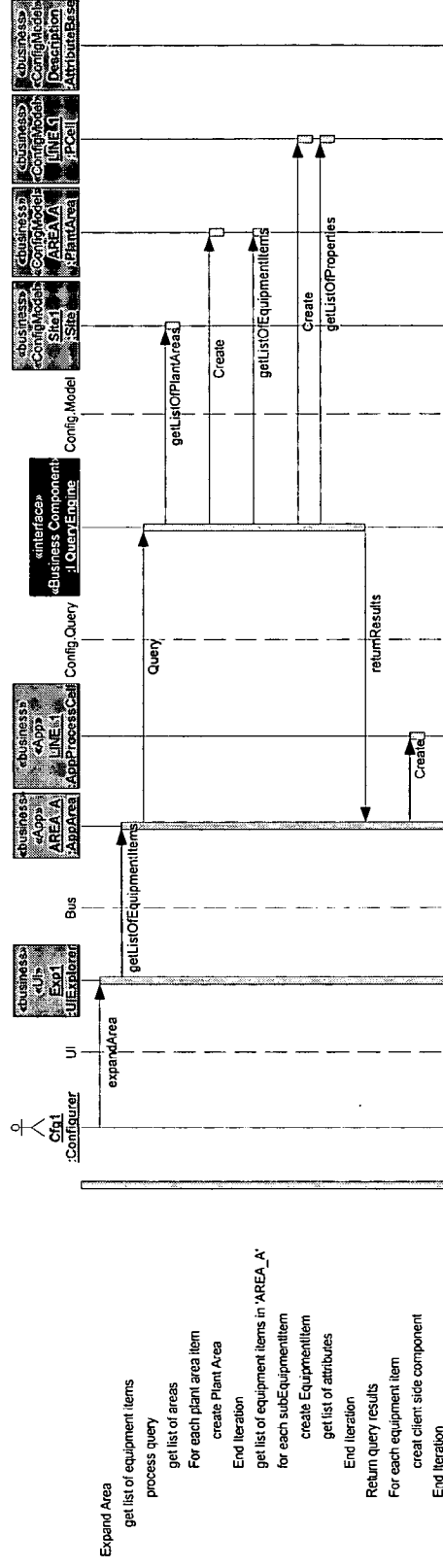
- 3- Query results returned

```
<?xml version="1.0" encoding="UTF-8"?>
< Results schema="1.0">
  <Site name="The Site">
    <PlantAreas>
      <PlantArea name="AREA_A">
        <SubEquipmentItems>
          <EquipmentItem name="LINE_1">
            <Properties>
              <type>PROCESS_CELL</type>
              <description> A Process Cell for LINE_1</description>
            </Properties>
          </EquipmentItem>
        </SubEquipmentItems>
      </PlantArea>
    </PlantAreas>
  </Site>
</Results>
```

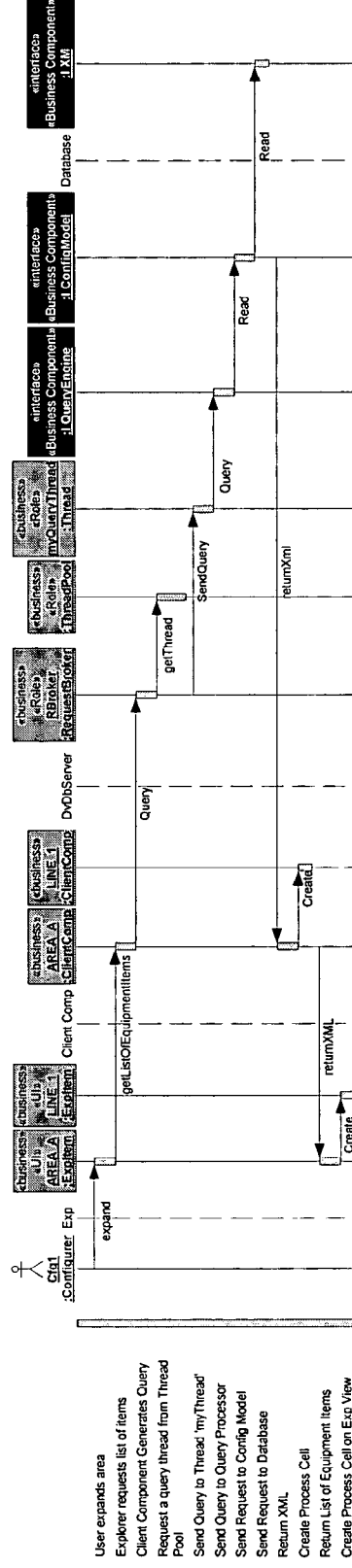
- 4- Explorer View Shows Process Cell 'LINE_1'

2.2.4 Component Sequence Diagram

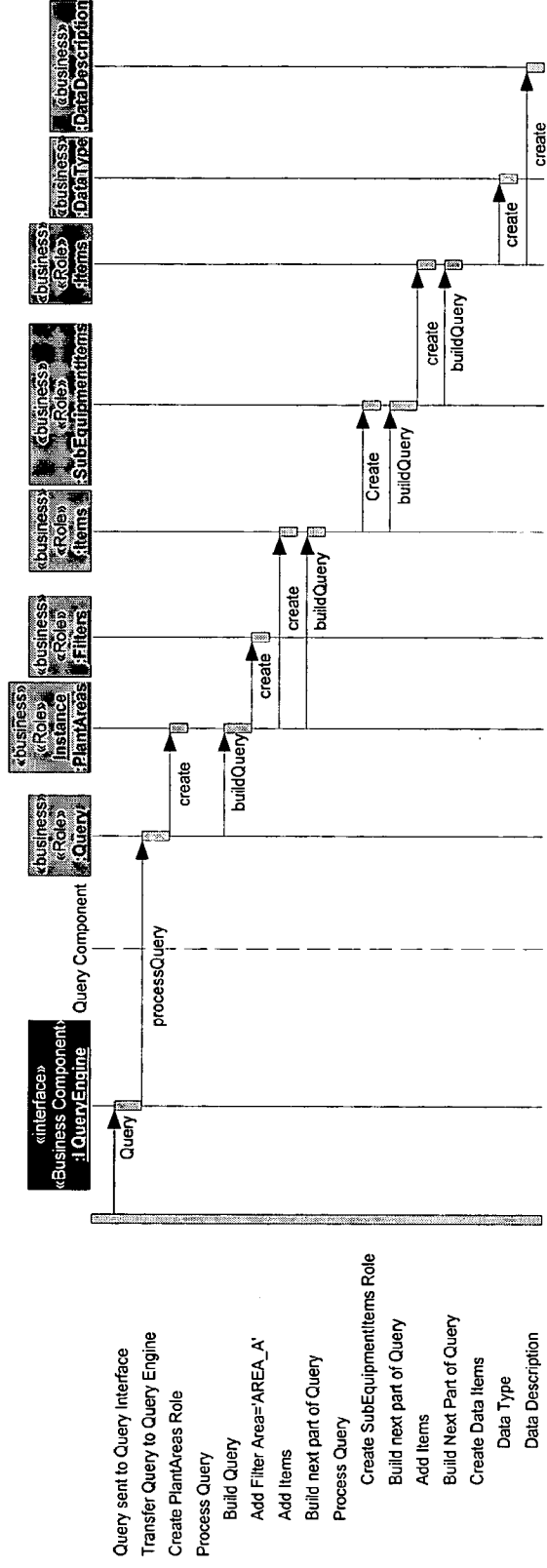
2.2.4.1 Expand Area 'AREA_A'



2.2.4.2 Expand Area 'AREA A' – DvDbServer



2.2.4.3 Expand Area 'AREA_A' – Query Engine



220

2.3 Expand Process Cell 'LINE_1'

2.3.1 Preconditions

- 1- Area 'AREA_A' expanded.

2.3.2 Postconditions

- 1- Process Cell 'LINE_1' expanded
- 2- Explorer shows Unit Module 'BOILER_A'

2.3.3 Scenario

- 1- User selects Process Cell 'LINE_1'
- 2- App generates query

Site.PlantAreas[name='AREA_A']().SubEquipmentItems[name='LINE_1']().
EquipmentItems(name,type,description).UnitModule.AssignedTo(name)

- 3- Query results returned

```
<?xml version="1.0" encoding="UTF-8"?>
<Results schema="1.0">
  <Site name="The Site">
    <PlantAreas>
      <PlantArea name="AREA_A">
        <SubEquipmentItems>
          <EquipmentItem name="LINE_1">
            <Properties>
              <type>PROCESS_CELL</type>
              <description> A Process Cell for LINE_1</description>
            </Properties>
            <SubEquipmentItems>
              <EquipmentItem name="BOILER_A">
                <Properties>
                  <type>UNIT</type>
                  <description> A Unit</description>
                  <assignedTo name="CTLR1"/>
                </Properties>
              </EquipmentItem>
            </SubEquipmentItems>
          </EquipmentItem>
        </SubEquipmentItems>
      </PlantArea>
    </PlantAreas>
  </Site>
</Results>
```

- 4- Explorer View shows Unit 'BOILER_A'

2.3.4 Component Sequence Diagram

2.4 Create Process Cell 'LINE_2' under Area 'AREA_A'

2.4.1 Preconditions

- 1- Area 'AREA_A' expanded.
- 2- Explorer positioned on Area 'AREA_A'

2.4.2 Postconditions

- 1- Process Cell 'LINE_2' created under Area 'AREA_A'
- 2- Explorer shows Process Cell 'LINE_2' with desc 'Cooling Water System'

2.4.3 Scenario

- 1- User right clicks Area 'AREA_A' and selects Create Process Cell
- 2- User enters name 'LINE_2' and sets description to 'Cooling Water System'
- 3- App creates command script¹

```
Import System
Import DeltaV.Config.Command
Function Command()
{
    Var areaContext=Item2.Open("Site.PlantAreas[name='AREA_A']");
    Var pcellContext=areaContext.execute("createProcessCell(name='LINE_2')");
    pcellContext.execute("setProperties(description='Cooling Water System')");
}
```

- 4- Command engine executes script (*details shown in sequence diagram*)
- 5- Command engine returns success
- 6- App generates query to load equipment items

```
Site.PlantAreas[name='AREA_A'](name).SubEquipmentItems(name,type,description)
```

- 7- Query results returned

```
<?xml version="1.0" encoding="UTF-8"?>
<Results schema="1.0">
  <Site name="The Site">
    <PlantAreas>
      <PlantArea name="AREA_A">
        <SubEquipmentItems>
          <EquipmentItem name="LINE_1">
            <Properties>
              <type>PROCESS_CELL</type>
              <description> A Process Cell for LINE_1</description>
            </Properties>
          </EquipmentItem>
          <EquipmentItem name="LINE_2">
            <Properties>
              <type>PROCESS_CELL</type>
              <description>Cooling Water System</description>
            </Properties>
          </EquipmentItem>
        </SubEquipmentItems>
      </PlantArea>
    </PlantAreas>
  </Site>
</Results>
```

¹ The command script is JScript

² Item is written in C#

```

        </PlantArea>
    </PlantAreas>
</Site>
</Results>

```

- 8- Explorer View shows Process Cell 'LINE_2'

2.4.4 Component Sequence Diagram

2.5 Create DI Card under Controller 'CTRL1'

2.5.1 Preconditions

- 1- Controller 'CTRL1' expanded.
- 2- No card exists in Slot 3

2.5.2 Postconditions

- 1- DI Card created under Controller 'CTRL1'
- 2- Explorer View shows card 'C03' of type 'DI Card, 8 Ch., 120VAC, Dry Contact'

2.5.3 Scenario

- 1- User right clicks on Controller 'CTRL1' and selects 'Create Card'
- 2- App Generates Query

```

Site.PCN.ACN.Node[name='CTRL1'].IO[index=1].validCardTypes(defname,Uname,revision)
:
Site.CardDefns[name=defname].channel(position).validChannelTypes(defname,Uname,revision)

```

<< NOTE - need to issue the first Query and then use the results to generate the 2nd query...not shown here >>

- 3- Query Results Returned
- 4- Selection Dialog Populated
- 5- User Selects DI Card, 8 Ch., 120VAC, Dry Contact, Slot=3
- 6- App creates command script³

```

Import System
Import DeltaV.Config.Command
Function Command()
{
    Var controllerContext = Item.Open("Site.pcn.acn.node[name='CTRL1'].IO[index=1]");
    Var cardContext = controllerContext.execute("createCard(slot=3,defname='DI_8CH_115VAC_DCT');

    // repeat for all eight channels
    For (ch=1;ch <= number_of_channels; ch++)
    {

```

³ The command script is JScript

```

        Var channelConext = Card.execute("createChannel(channel='1'type=' DI_CHAN');
        channelContext.execute("createDST(dstName="**");
    )
    ...
}

```

"OR", the preferred mechanism, have the server side create the channels. In this case the script would simply be:

```

Import System
Import DeltaV.Config.Command
Function Command()
{
    Var controllerContext = Item.Open("Site.pcn.acn.node[name='CTLR1'].IO[index=1]");
    Var cardContext = controllerContext.execute("createCard(slot=3,defname='DI_8CH_115VAC_DCT");
}

```

- 7- Command engine executes script (*details shown in sequence diagram*)
- 8- Command engine returns success
- 9- App generates query to load node items

```
Site.pcn.acn.node[name='CTLR1'].IO[index=1].card(name).channel(name,description,dst);
```

- 10- Query results returned

```

<?xml version="1.0" encoding="UTF-8"?>
< Results schema="1.0">

```

...need an example...

```
</Results>
```

- 11- Explorer View shows Card 'C03' under Controller 'CTLR1'

2.5.4 Component Sequence Diagram

2.6 Convert FB 'AI1' from defn 'AI' to 'FFAI' (w/o updates)

2.6.1 Preconditions

- 1- Module 'FLOW_METER_1' contains FBU 'AI1' from definition 'AI'
- 2- User has explorer view open on module 'FLOW_METER_1'

2.6.2 Postconditions

- 1- Module 'FLOW_METER_1' contains FBU 'AI1' from definition 'FFAI'

next items come into play if there is view up that contains these items, i.e., something has to have registered for updates

- 2- *Parameters disappear (some)*

- 3- *Lines disappear (some)*
- 4- *Bindings disappear (AI1 initially bound to 'CTLR1/IO1/C01/CH1')*
- 5- *Over-rides are maintained where applicable*

2.6.3 Scenario

- 1- User selects FBU 'AI1' and selects choice 'Convert to Fieldbus'
- 2- App creates commands

```

Import System
Import DeltaV.Config.Command
Function Command()
{
    Var fbu=Item.Open("Site.modules[name='FLOW_METER_1'].blocks[name='AI1']");
    fbu.execute("setblockdefinition('FFAI')");
}

```

- 3- Command engine executes script (*details shown in sequence diagram*)
- 4- Command engine returns success
- 5- App generates query to load items

```
Site.modules[name='FLOW_METER_1'].blocks(..);
```

>> loading all of the details in the blocks since there could be many changes (including parameter deletions, renames, and defn changes).

>> also, need to re-run this scenario with a Control Studio type view to show the lines, parameter changes (disappearing/adding), and parameter over-rides

- 6- Query results returned

```

<?xml version="1.0" encoding="UTF-8"?>
< Results schema="1.0">

...need to complete example...

</Results>

```

- 7- Explorer View shows updated definition

2.6.4 Component Sequence Diagram

2.7 Convert FB 'AI1' from defn 'AI' to 'FFAI' (w/ updates)

For this scenario we simplified the hierarchy – the modules 'VALVE_1' and 'FLOW_METER_1' have been moved up directly under Area 'AREA_A'.

There were side conversations here on how to approach this whole area of updates. It was agreed that providing the users with preferences would be ideal:

- 1- *Refresh Immediately*
- 2- *Refresh periodically (time specified by user)*
- 3- *Flag changes but don't refresh, manually refresh*
- 4- *Do not flag changes, manually refresh*

2.7.1 Preconditions

- 1- Module 'FLOW_METER_1' contains FBU 'AI1' from definition 'AI'
- 2- Module 'FLOW_METER_1' contains FBU 'CALC1' with an internal reference '^/AI1/OUT.CV'.
- 3- Module 'VALVE_1' contains external reference parameter 'P1' referencing 'FLOW_METER_1/AI1/OUT.CV'.
- 4- User has explorer view open on module 'FLOW_METER_1'

2.7.2 Postconditions

- 1- Module 'FLOW_METER_1' contains FBU 'AI1' derived from definition 'FFAI'.
- 2- Internal and external references described in Preconditions are maintained.
- 3- Binding to 'CTRL1/IO1/C01/CH1/FIELD_VAL_PCT' replaced by 'PDT1.FAI1'

2.7.3 Scenario

- 1- User selects Area 'AREA_A'
- 2- App generates query

```
updateContext=(Site.PlantArea[name='AREA_A'].(handle)TopLevelModules(handle, name, type, desc)
```

- 3- Query results

```
<?xml version="1.0" encoding="UTF-8"?>
< Results schema="1.0">
  <Site name="The Site">
    <PlantAreas>
      <PlantArea name="AREA_A">
        <TopLevelModules handle="h1">
          <ModuleItem name="FLOW_METER_1" handle="h2">
            <Properties>
              <type>MODULE</type>
              <description>A module</description>
            </Properties>
          </ModuleItem>
          <ModuleItem name="VALVE_1" handle="h3">
            <Properties>
              <type>MODULE</type>
              <description>Another module</description>
            </Properties>
          </ModuleItem>
        </TopLevelModules>
      </PlantArea>
    </PlantAreas>
  </Site>
```

</Results>

- 4- App constructs update list

```
collection 'h1'
  module 'h2'
  module 'h3'
```

- 5- App generates query

```
Site.Modules[name='FLOW_METER_1'].blocks(handle, name, type, defname, algtype)
```

- 6- Query Results

```
<?xml version="1.0" encoding="UTF-8"?>
< Results schema="1.0">
  <blocks>
    <BlockItem name="AI1" handle="h4">
      <Properties>
        <type>AI</type>
      </Properties>
    </BlockItem>
    <BlockItem name="CALC1" handle="h5">
      <Properties>
        <type>CALC</type>
      </Properties>
    </BlockItem>
  </blocks>
</Results>
```

- 7- App builds tree

- 8- User selects FBU 'AI1' and selects choice 'Convert to Fieldbus'

- 9- User browses to 'PDT1/AI1' under Node 'CTLR1/IO1/C02/P01'

- 10- App creates commands

```
Import System
Import DeltaV.Config.Command
Function Command()
{
  Var fbu=Item.Open("Site.modules[name='FLOW_METER_1'].blocks[name='AI1']");
  fbu.execute("setblockdefinition('FFAI')");
}
```

- 11- Command engine executes script (*details shown in sequence diagram*)

- 12- Command engine returns success

- 13- Db detects change to FBU 'AI1'

- 14- Db adds all handles it associated with 'AI1' to the update list

- 15- Db sends update list to Config.Updates (part of DvDbServer)

- 16- Config.Updates locates client 'MyClient' in update list

- 17- Config.Updates sends update notification to Client 'MyClient'

```
h4 – associated with FLOW_METER_1/AI1
h5 – associated with CTLR1/IO1/C01/CH1/FIELD_VAL_PCT
h6 – associated with PDT1/AI1
```

18- Client 'MyClient' receives update and processes updates

Client 'MyClient' finds AppObject 'FLOW_METER_1/AI1' using handle 'h4' and marks dirty
Client 'MyClient' finds AppObject 'CTRL1/IO1/C01/CH1/FIELD_VAL_PCT' using handle 'h5' and marks dirty

Client 'MyClient' finds AppObject 'PDT1/AI1' using handle 'h6' and marks dirty

19- Client 'MyClient' update thread finds dirty bits

20- Client 'MyClient' generates query

```
Site.Modules[name='FLOW_METER_1']blocks[name='AI1'](handle. name, type, desc)
```

21- Query results returned

```
<?xml version="1.0" encoding="UTF-8"?>
< Results schema="1.0">
  <blocks>
    <BlockItem name="AI1" handle="h4">
      <Properties>
        <type>FFAI</type>
      </Properties>
    </BlockItem>
    <BlockItem name="CALC1" handle="h5">
      <Properties>
        <type>CALC</type>
      </Properties>
    </BlockItem>
  </blocks>
</Results>
```

22- Client updates appObject 'AI1'

23- Client updates update map (in this case there is nothing to do)

24- Repeat for 'CTRL1/IO1/C01/CH1/FIELD_VAL_PCT', dst1, and 'PDT1/AI1'

2.7.4 Component Sequence Diagram

2.8 Change Process Cell Name from 'LINE_1' to 'UTIL' (w/ updates)

2.8.1 Preconditions

- 1- Area 'AREA_A' not expanded.
- 2- Process Cell named 'LINE_1'

2.8.2 Postconditions

- 1- Process Cell named 'UTIL'

2.8.3 Scenario

- 1- User selects Area 'AREA_A'

- 2- App generates Query

```
Site.PlantArea[name='AREA_A'].(handle)EquipmentItems(handle,name,type)
```

- 3- Query Results

```
<?xml version="1.0" encoding="UTF-8"?>
<Results schema="1.0">
  <EquipmentItems handle="h1">
    <EquipmentItem name="LINE_1" handle="h2">
      <Properties>
        <type>PROCESS CELL</type>
      </Properties>
    </EquipmentItem>
  </EquipmentItems>
</Results>
```

- 4- User selects Process Cell 'LINE_1' and renames to 'UTIL'

- 5- App generates command

```
Import System
Import DeltaV.Config.Command
Function Command()
{
  Var context=Item.Open("Site.EquipmentItems{name='LINE_1'}");
  context.execute("renameTo('UTIL')");
}
```

- 6- Command engine executes script and returns DB_SUCCESS

- 7- Db detects changes to Process Cell 'UTIL'

- 8- Db adds handle 'h2' to change list, contains hint('rename', 'LINE_1', 'UTIL')

- 9- Db sends update list for client 'MyClient'

- 10- DvDbServer locates 'MyClient' in update map

- 11- DvDbServer sends update notification with 'h2' to Client 'MyClient'

- 12- Client 'MyClient' receives update list, marks items dirty

- 13- Client 'MyClient' update thread finds dirty bits

- 14- Client 'MyClient' generates query

```
Site.PlantArea[name='AREA_A'].(handle)EquipmentItems(handle,name,type)
```

...alternative...

- 15- Client 'MyClient' compares existing to new name. Since the client already processed the update, i.e. when it did the rename, it does not execute a new query sequence.

2.8.4 Component Sequence Diagram

2.9 Change name of FBU 'AI1' to 'FLOW1' (w/updates)

In this scenario we expand out the hierarchy to illustrate how the update handles are allocated and later incorporated into the updates. Only the Control Strategy updates are considered in the discussion.

2.9.1 Preconditions

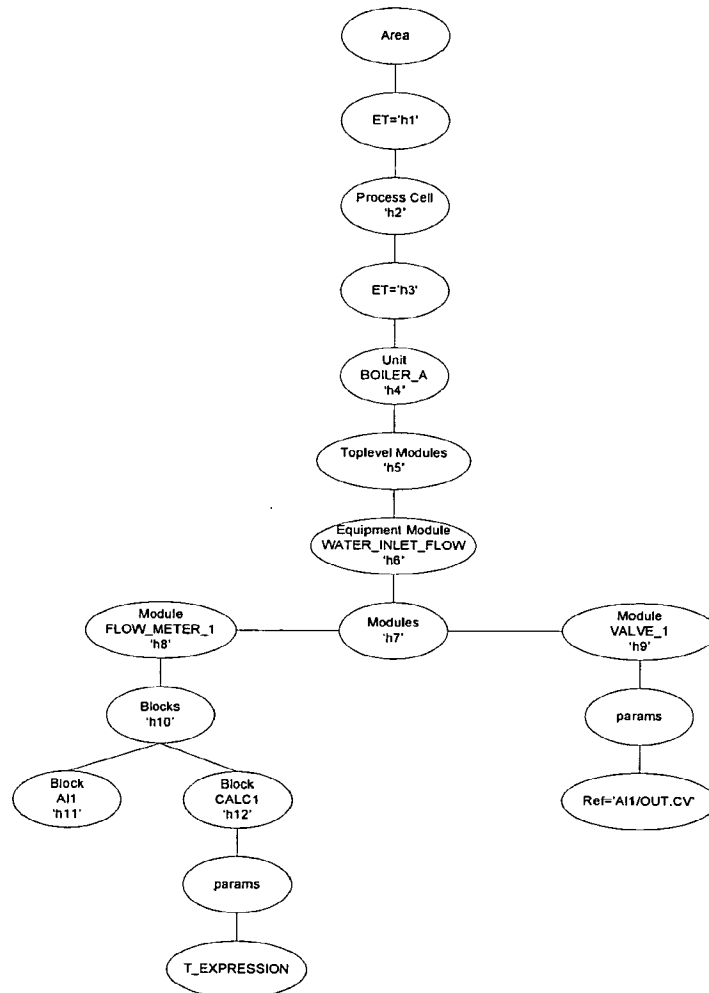
- 1- Area 'AREA_A' not expanded.
- 2- FBU 'AI1' exists in Module 'FLOW_METER_1'

2.9.2 Postconditions

- 1- Area 'AREA_A' fully expanded
- 2- FBU 'AI1' renamed to 'FLOW1'

2.9.3 Scenario

The drawing below provides background information on the scenario:



- 1- User selects Area 'AREA_A' and expands it
- 2- App generates series of sequence as user expands hierarchy

```

Site.PlantAreas[name='AREA_A'].(handle).EquipmentItems(handle,name,type)
Site.EquipmentItems[name='LINE_1'].(handle).EquipmentItems(handle,name,type)
Site.EquipmentItems[name='BOILER_A'].(handle).TopLevelModules(handle,name,type)
Site.TopLevelModules[name='WATER_INLET_FLOW'].(handle).TopLevelModules(handle,name,
type)
Site.TopLevelModules[name='FLOW_METER_1'].(handle).Blocks(handle,name,type)
Site.TopLevelModules[name='VALVE_1'].(handle).Attributes(name,type,ref)
Site.TopLevelModules[name='FLOW_METER_1'].(handle).Blocks[name='CALC1'].Attributes(na
me,type,expression)

```

- 3- User selects FBU 'AI1' and renames it to 'FLOW1'
- 4- App generates command sequence

```

Import System
Import DeltaV.Config.Command
Function Command()
{
    Var context=Item.Open("Site.TopLevelModule[name='FLOW_METER_1'].Blocks[name='AI1']");
    context.execute("renameTo('FLOW1')");
}

```

NOTE – alternatively, we also discussed being able to support this same transaction using handles as show below:

```

Function Command()
{
    Var context=Item.Open("Site.handles[handle='h11']");
    context.execute("renameTo('FLOW1')");
}

```

- 5- Command Processor executes script
- 6- Db detects changes to FBU 'FLOW1'
- 7- Db adds handles 'h11,h12,h8,h9' to change list, contains hint('rename', 'LINE_1', 'UTIL')
- 8- Db sends update list for client 'MyClient'
- 9- Client 'MyClient' receives update list, marks items dirty
- 10- Client 'MyClient' update thread finds dirty bits
- 11- Process update event 'h11' : Update(handle='h11', rename,'AI1','FLOW1')
 - >>rename 'AI1' to 'FLOW1'
 - Client renames FBU 'AI1' to 'FLOW1'
- 12- Process update event 'h12' : Update(handle='h12', assoicatedItem)
 - >> update on FBU 'CALC1'
 - Client generates Query
 - Site.Handles[handle='h12'].attributes(...)
 - Client updates parameters

- 13- Process update event 'h8' : Update(handle='h8', assoicatedItem)
 - >> update on module 'FLOW_METER_1'
 - Client generates Query
 - Site.Handles[handle='h8']
 - {
 - (handle) Blocks(handle,name,type);
 - attributes(...);
 - }
 - Client updates
- 14- Process update event 'h9' : Update(handle='h9', assoicatedItem)
 - >> update on module 'VALVE_1'
 - Client generates Query
 - Site.Handles[handle='h9']
 - {
 - (handle) Blocks(handle,name,type);
 - attributes(...);
 - }

2.9.4 Component Sequence Diagram

2.10 Find all Modules with parameter 'MSTATUS_MASK' with Bit 4 set in Plant Area 'AREA_A'

2.10.1 Preconditions

- 1- User has launched browser (i.e. Find utility)

2.10.2 Postconditions

- 1- Browser returns list of Modules meeting search/filter criteria

2.10.3 Scenario

- 1- User selects search / filter criteria to find all Modules in Area 'AREA_A' with parameter 'MSTATUS_MASK' with Bit 4 set
- 2- App generates command

```

Import System
Import DeltaV.Config.Command
Function Command()
{
    Var streamHandle = Item.OpenStream("Site");
}

```

- 3- Server executes command and returns Stream handle 'streamHandle'
- 4- App generates command

```

Import System
Import DeltaV.Config.Command

```

```

Function Command()
{
    streamHandle.execute(streamHandle,
    "Find(Site.PlantAreas[name='AREA_A']..modules.(name,type,path).attributes[name='MSTATUS_MASK'].Fields[name='CV', value & 0x1000 != 0]);
}

```

- 5- App reads stream
- 6- Server returns results

example...

```

<?xml version="1.0" encoding="UTF-8"?>
< Results schema="1.0">
    <Module name="VALVE_1">
        <Properties>
            <type>MODULE</type>

            <path>Site.PlantAreas[name='AREA_A'].EquipmentItems[name='LINE_1'].EquipmentItems[name='BOILER_A'].
EquipmentItems[name='FLOW_METER_1'].Modules[name='VALVE_1']</path>
        </Properties>
    </Module>
</Results>

```

- 7- Repeat 5 & 6 until server is complete – it then returns search complete status

2.10.4 Component Sequence Diagram

2.11 Find all Modules of type 'PLM' in Plant Area 'AREA_A'

2.11.1 Preconditions

- 1- User has launched browser (i.e. Find utility)

2.11.2 Postconditions

- 1- Browser returns list of Modules meeting search/filter criteria

2.11.3 Scenario

- 1- User selects search / filter criteria to find all Modules in Area 'AREA_A' of Type 'PLM'
- 2- App generates command

```

Import System
Import DeltaV.Config.Command
Function Command()
{
    Var streamHandle = Item.OpenStream("Site");
}

```

- 3- Server executes command and returns Stream handle 'streamHandle'

4- App generates command

```
Import System
Import DeltaV.Config.Command
Function Command()
{
    streamHandle.execute(streamHandle,
    "Find(Site.PlantAreas[name='AREA_A']..modules.(name,type,path)[type='PLM'])");
}
```

5- App reads stream

6- Server returns results

example...

```
<?xml version="1.0" encoding="UTF-8"?>
< Results schema="1.0">
  <Module name="PLM_1">
    <Properties>
      <type>MODULE</type>

      <path>Site.PlantAreas[name='AREA_A'].EquipmentItems[name='LINE_1'].EquipmentItems[name='BOILER_A'].
EquipmentItems[name='FLOW_METER_1'].Modules[name='PLM_1']</path>
    </Properties>
  </Module>
</Results>
```

7- Repeat 5 & 6 until server is complete – it then returns search complete status

NOTE – as part of the search the user needs to be able to cancel the search. If the user were to do this then the scenario would complete as follows:

```
User cancels search
App sends cancel to Server
Server aborts search
Server returns completion status 'Search Cancelled' back to the User
```

2.11.4 Component Sequence Diagram

3 Technical Architecture

3.1 Overview

The technical architecture delivers the technical components and frameworks that support what we are trying to build. The details of the technical architecture are covered in the Starburn Technical Architecture document.

3.2 Component Architecture

The drawing below illustrates the initial component map after the exercises executed above. Packages have been added to represent the logical components. Most components have an interface added to provide services.

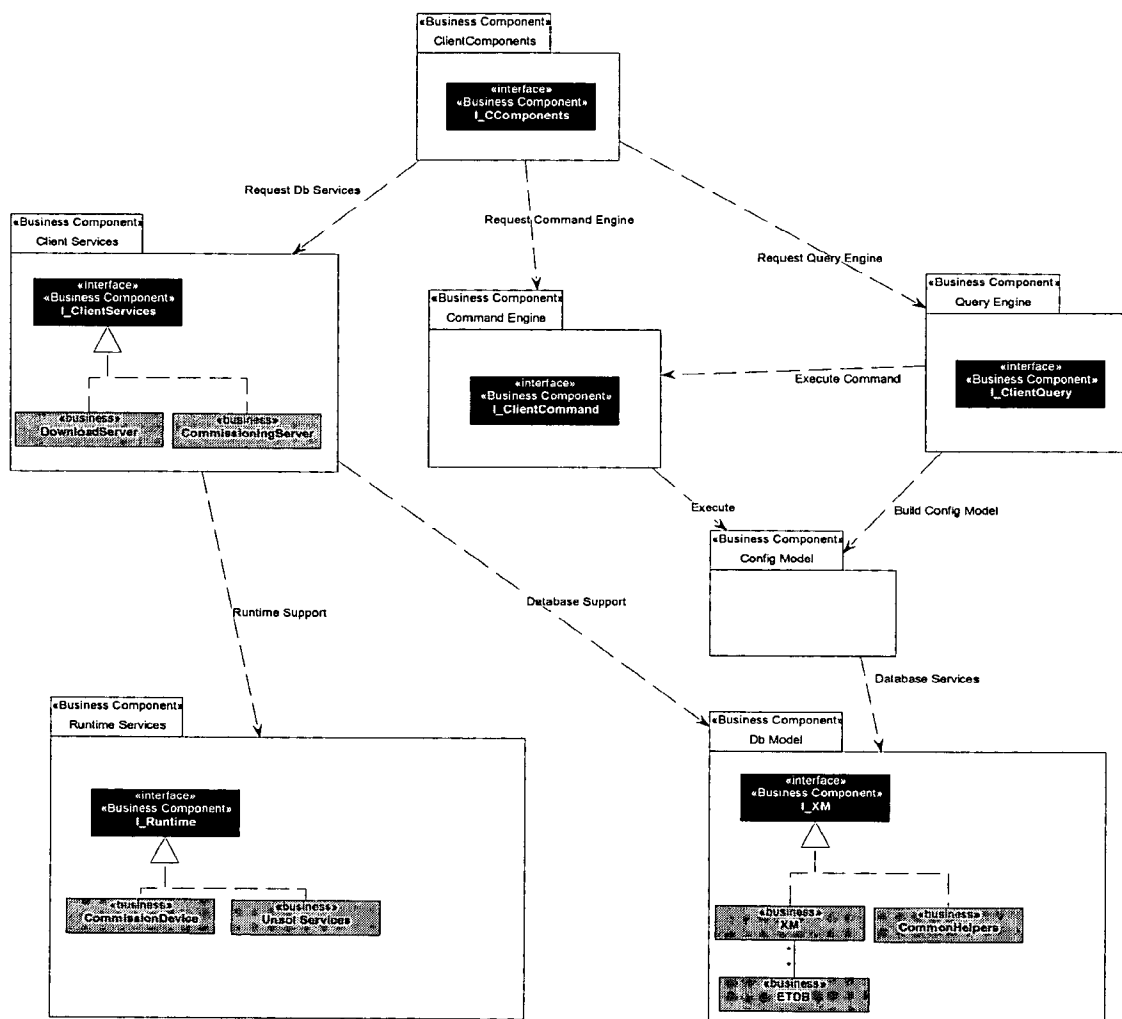


Figure 2. Component Architecture

3.3 Subsystems

Subsystems provide a high-level view of how the architecture realizes Use Cases without going into the low-level details. The concept of subsystems sometimes carries through to implementation in which case real components implement the subsystem.

Subsystem Name	Subsystem Description
Operator Workspace	Provides a controlled environment for Process Graphics, Batch Operator Interfaces, Advanced Control Applications, Alarm Banner, Alarm Summary, Faceplates, Detail displays, History Clients, etc.
Engineering Workspace	Provides a controlled environment for Configuring Process Graphics, Batch, Advanced Control, Alarms, Displays, Devices, History, etc.
Process Graphics	Provide user defined views and operator interfacing onto the system architecture.
UI	Provides Specific Controls, Frameworks for application developers, UI design guidelines, etc
Configuration	Provides configuration services for both connected and disconnected clients. Connected clients include applications to configure control strategies, displays, IO, user accounts, manage configuration, load licenses, etc. Disconnected clients are used for bulk editing the system, for interfacing to external tools such as INtools, and for addressing customer specific requirements with respect to Recipe Management, etc. Also includes services for Versioning and Audit Trail.
Communication	Provides communication services for control, debugging, configuration, alarm & alerts, operator interfaces, etc.
Data Services	Provides low level access to the system. This consists of things such as the Custom Function Block interface for interfacing foreign DCS systems (e.g. Bailey, Teleperm),
Session	Provide support for user logon, session management, secure access to the system, and span-of-control.
Events / Alarm / Alert Services	Allows monitoring and notification of significant system states, acknowledgements, and priority calculations.
Licensing	Provide services to enable functionality on the system using licensing.
Diagnostics	Interacts with all of the subsystems to collect diagnostic and alert information.
System Administration Services	Provides services for installing applications and hot fixes. Also provides support for remotely monitoring and supporting systems.
History	Provides access to continuous, event, and batch data.
Control	Provides continuous, sequencing, batch, recipe, process, safe control.
PIO	Provides access to all of the IO – including conventional IO, Fieldbus, Profibus, DevicNet, Serial, ASi, Remote IO, RS3 IO, Provox IO, Safe IO, etc.
Report Writer	Provides reporting services for alarm/alert information, document configuration, etc.

4 Configuration Applications and Supporting Frameworks

4.1 Overview

The configuration applications and their supporting frameworks are illustrated below⁴.

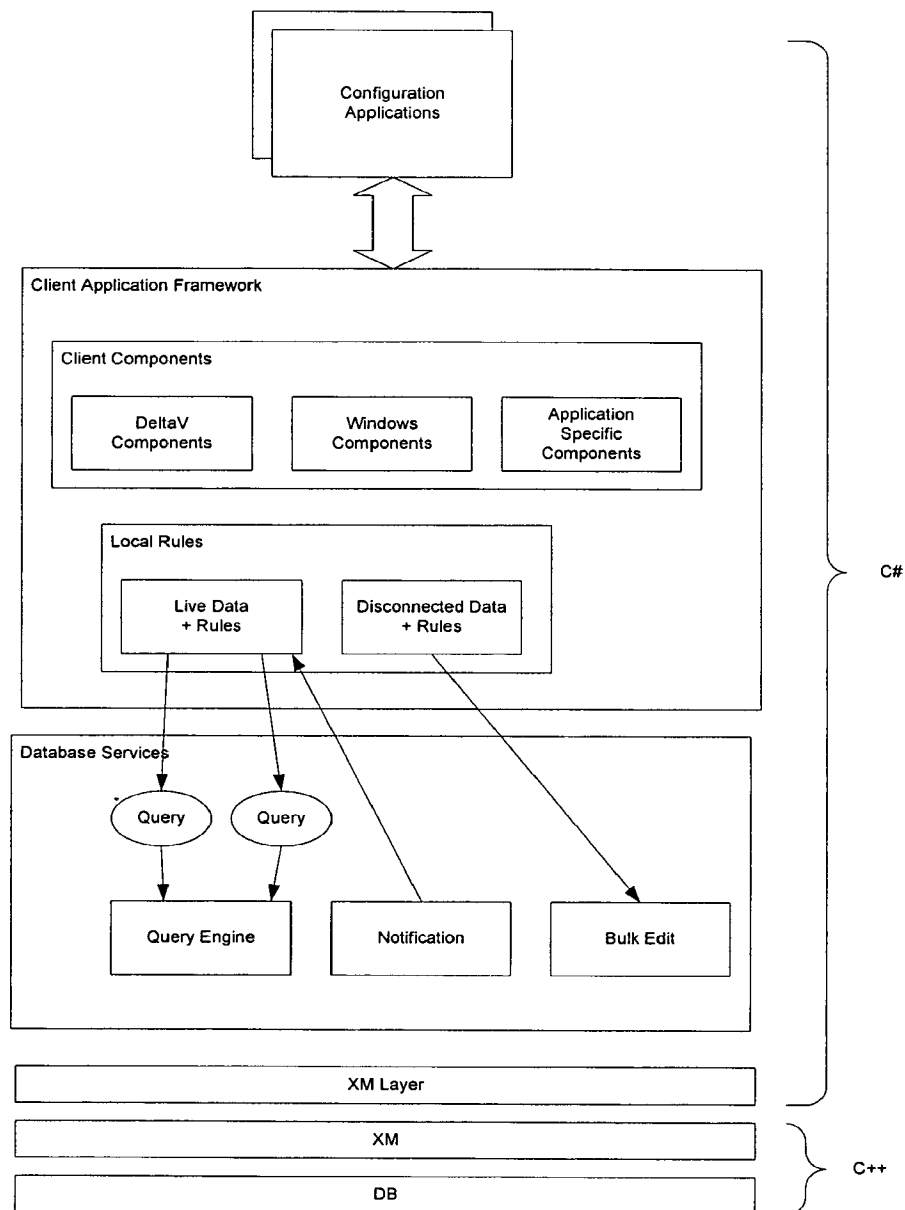


Figure 3. Configuration Architecture

The model is expanded further in the drawing below.

⁴ Supporting subsystems, such as the communications, are not shown.

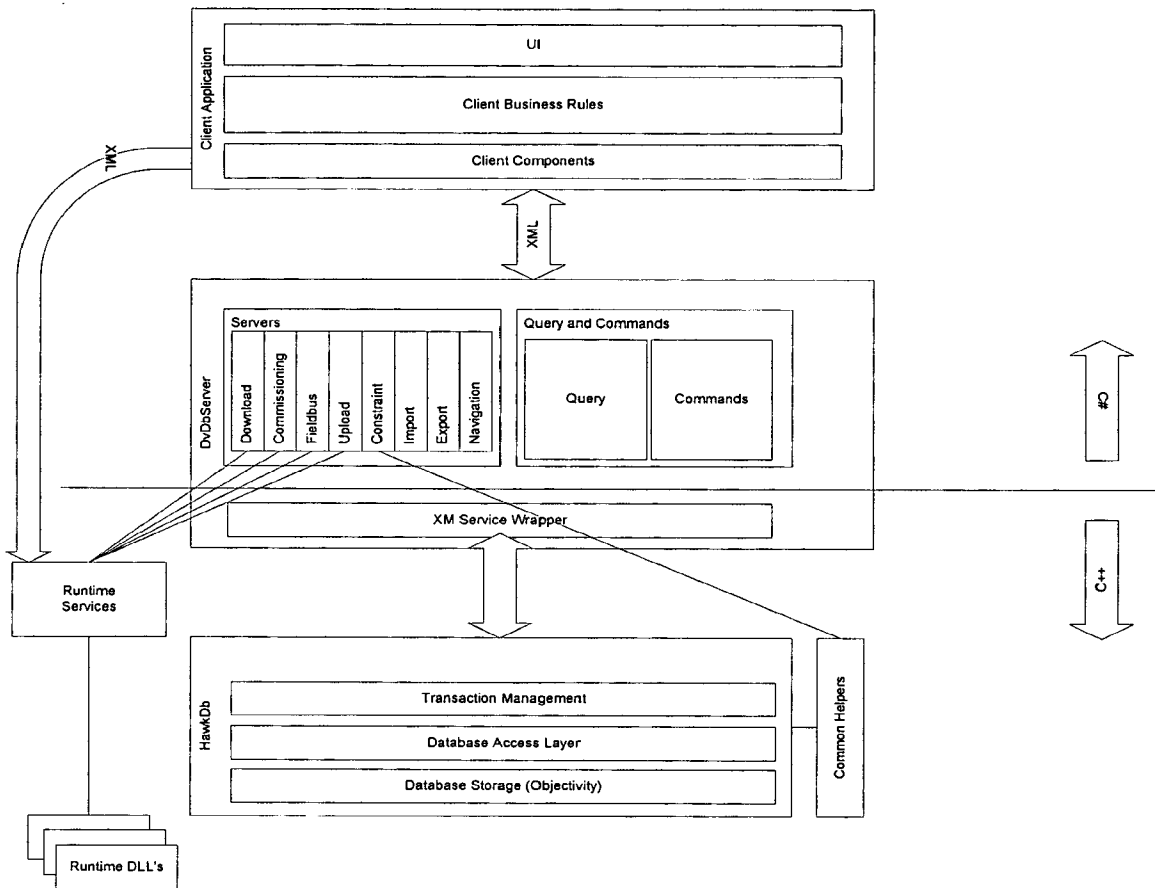


Figure 4. Deployment View of Configuration Framework

The Client and Server Item Caches would have similar roles to the existing Lightweight layers but would be re-implemented a) to use Xml interfaces rather than DCOM and b) to push more functionality into frameworks.

The DB load/save component would translate between Xml and XM calls.

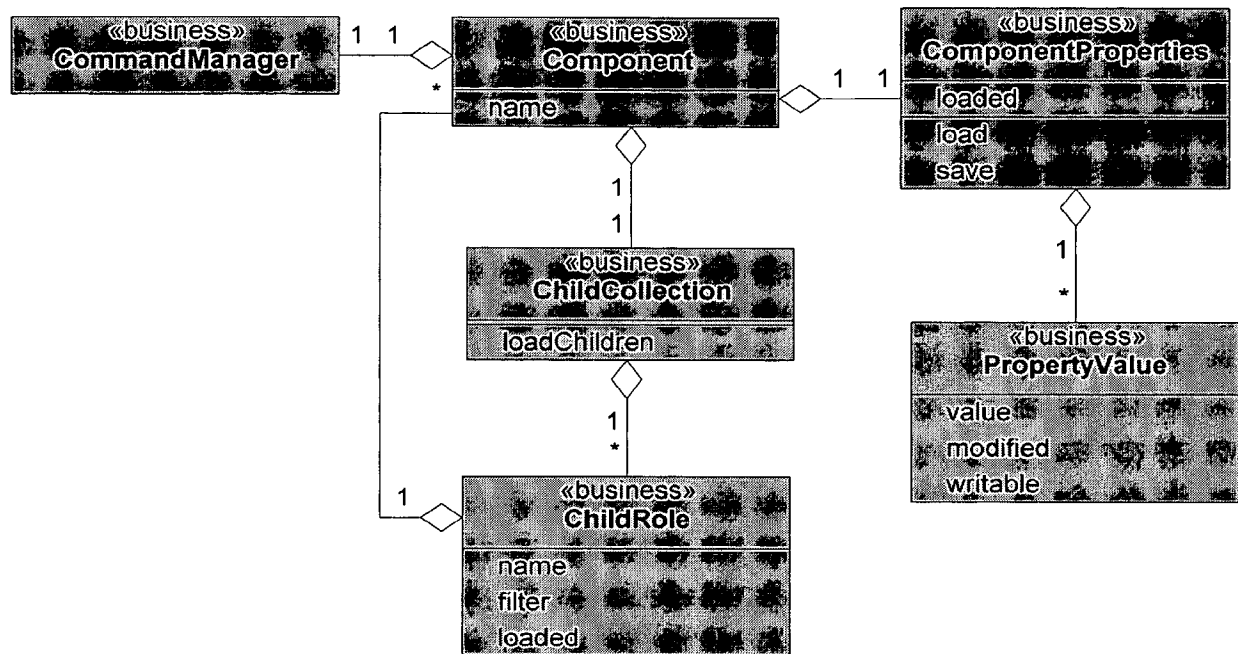
The load capability would be divided into loading of simple properties, whole items and special properties (that require significant Db processing and hence time).

XM Services means things like Download, Verification and Import/Export that invoke considerable processing within the database, as used by the various Server legs in the DvDbServer.

The XM Service Wrapper is merely a set of C# classes that wrap XM calls.

4.2 Client Component Framework

4.2.1 Object Model



The core of the framework is Component. It is broken down into 3 aspects, properties, children and commands.

4.2.2 Properties

Each Component contains a properties object. Properties can be loaded and saved independently of the component. Concrete subclasses of Properties contain members of type PropertyValue, this object encapsulates the value, whether it can be modified and whether it has been modified.

On save; the properties that have been modified are sent back to the server.

4.2.3 Children

ChildCollection aggregates a series of ChildRoles and implements loadChildren. Roles can be loaded individually or en masse. When a role is loaded a filter can be applied. The properties of the child components can optionally be loaded at the same time.

LoadChildren is called with a list of role descriptors that define exactly what is to be loaded. The framework is designed such that it can produce one query based on the supplied load criteria.

Query results are merged with any previously loaded children. If more roles are request these are simply added. An existing role can be requeried with a different filter. In this event members of the role common to both filtered lists are preserved, while additions and deletions are made.

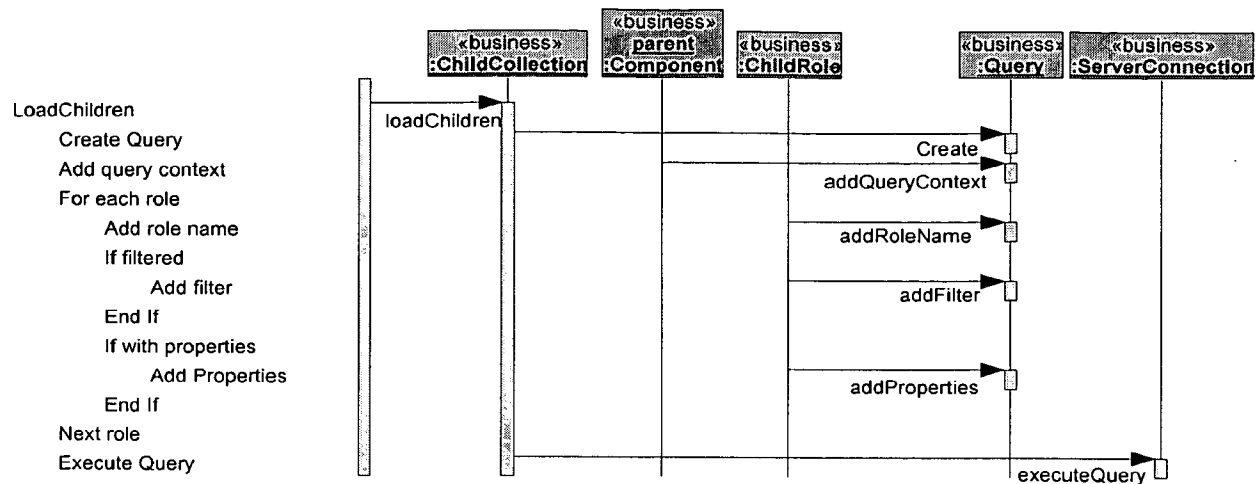


Figure 5 Client Components - LoadChildren

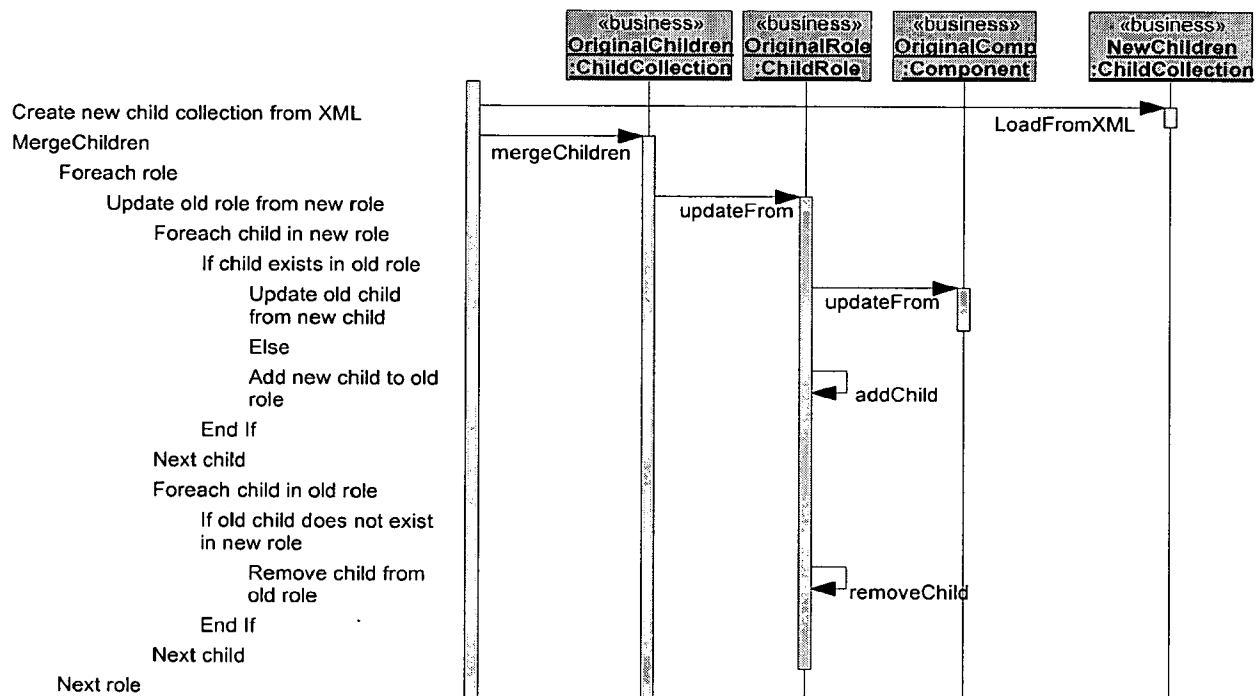


Figure 6 Client Components - MergeChildren

4.2.4 Commands

Each concrete subclass of Component comes with a CommandManager derived class. This class defines a series of methods that can insert the relevant commands into a command script.

A CommandBuilder class is used to create a complete command script that can be sent to the server. This allows a series of commands to be composed into one script.

Commands are given membership to one or more groups. Grouping is aimed at assisting the UI in knowing which commands are valid in certain situations.

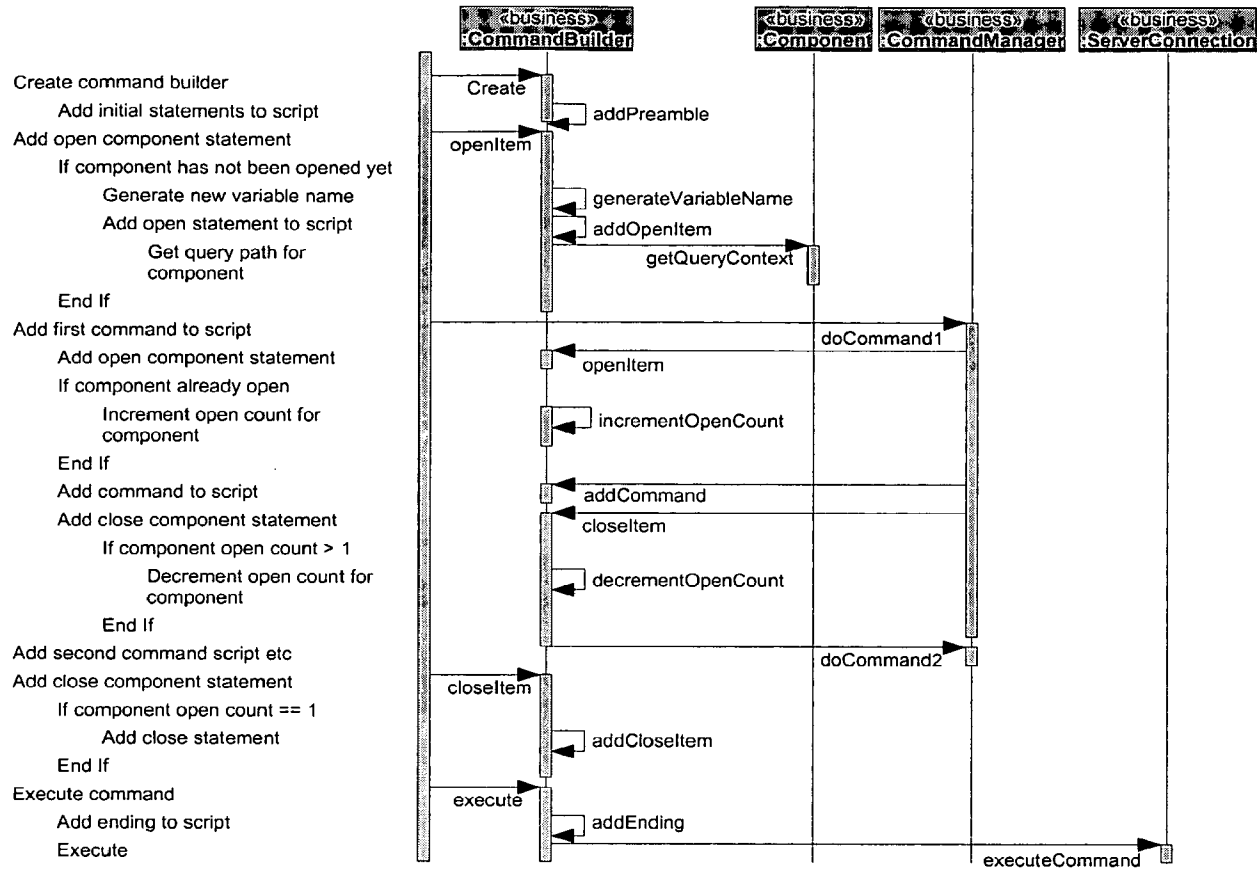


Figure 7 Client Components - Building a Command Script

4.3 Client/Server

4.3.1 Connectivity

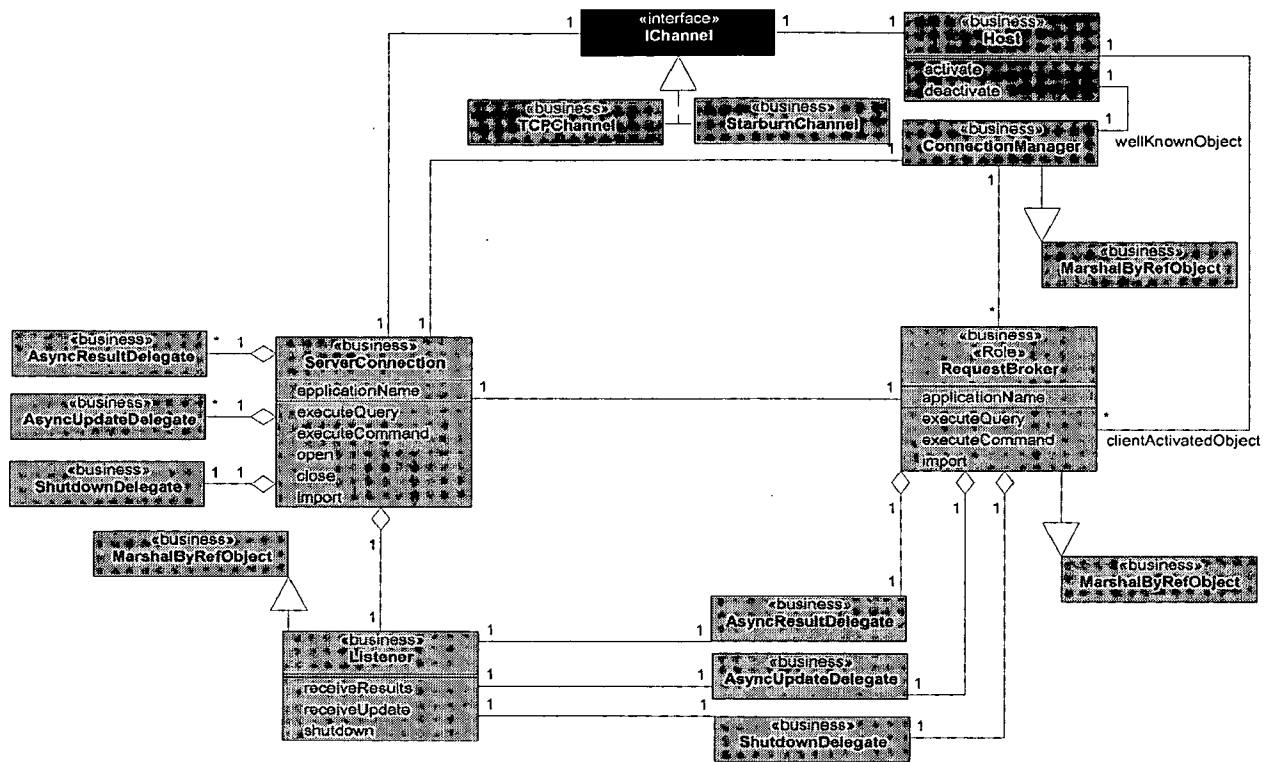


Figure 8 Client Server Connection

ServerConnection encapsulates the client side connection entity. It is used to establish a connection with the server and to send and receive information. The server equivalent for the ServerConnection class is RequestBroker. This represents the client application in the server.

ConnectionManager aggregates RequestBrokers and provides methods for interrogating and manipulating them from a client application. Host encapsulates the startup and shutdown code necessary to publish the .net remotable objects.

Delegates are used to send back asynchronous messages from the server to the client. The results from asynchronous queries and commands come through AsyncResultDelegate. Database update notifications come through AsyncUpdateDelegate. The server indicates to the client that it will be shutting down through ShutdownDelegate.

Client/Server communication is established through a .net remoting channel. In addition to the standard tcp and http channels there will be a starburn channel. The starburn channel will use the DeltaV communication subsystem to reliably communicate information within a DeltaV system and between DeltaV zones.

4.3.2 Threading

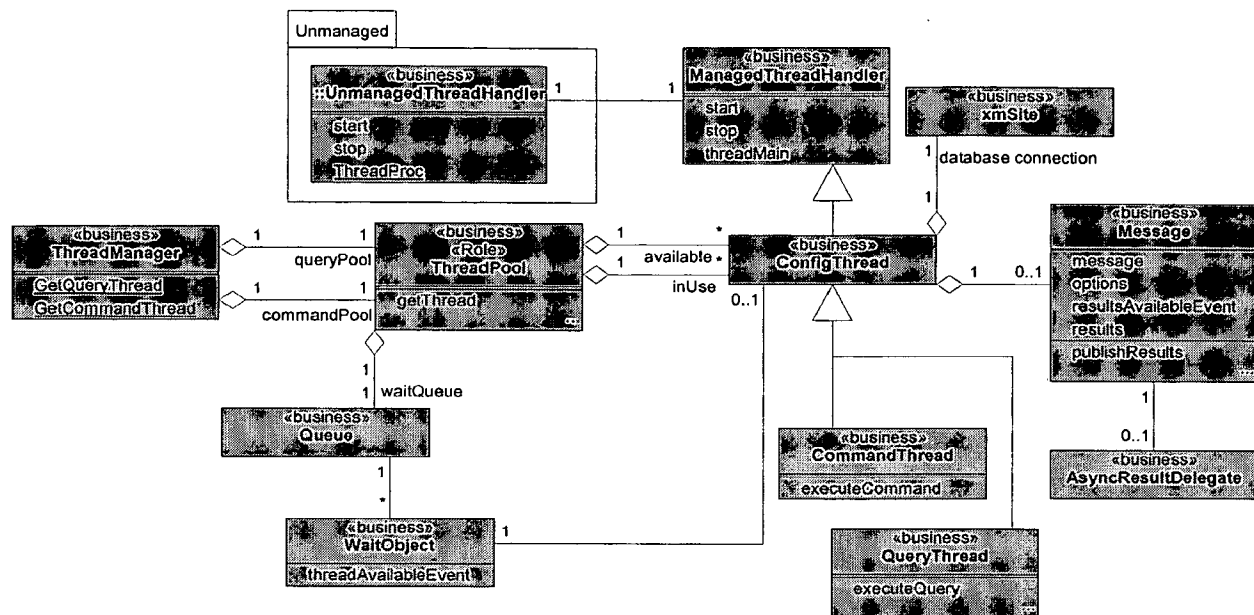


Figure 9 Server Threading Model

ThreadManager holds onto 2 static references to ThreadPool. One for query (read) threads the other for command (write) threads. A ThreadPool manages a number of ConfigThreads. At any one time some threads could be in use while others are available.

If all the threads in a pool are in use the next request to getThread will construct and enqueue a WaitObject. When a thread completes its current activity it checks the wait queue. If a client is waiting that thread is given to the client. Otherwise the thread returns to the list of available threads.

ConfigThread straddles 2 OS threads, the thread on which the request came in and the thread holding the database connection. All database access may be done on the latter thread. Further more the database thread may be a dedicated OS thread. The message can come in on any OS thread.

To ensure the database calls into a database connection always occur on the same OS thread we do not use managed threads. Instead, we implement dual managed and unmanaged thread classes. ConfigThread derives from ManagedThreadHandler which maintains a pointer to an UnmanagedThreadHandler. The latter class is able to start a real OS thread. The OS thread procedure simply executes the ManagedThreadHandler's threadMain. This method sits waiting for a 'message ready' event.

When a request is to be processed a MessageObject is constructed and store on the ConfigThread. The database thread is signaled to indicate a message is ready. Once the database thread has completed the task it stores the results on the MessageObject and instructs it to publish the results. If the request was sent synchronously the original thread on which the request arrived is signaled to indicate the results are available. This thread reads the results and returns to the client. If the request was sent asynchronously the message will have an associated AsyncResultDelegate which it can use to publish the results.

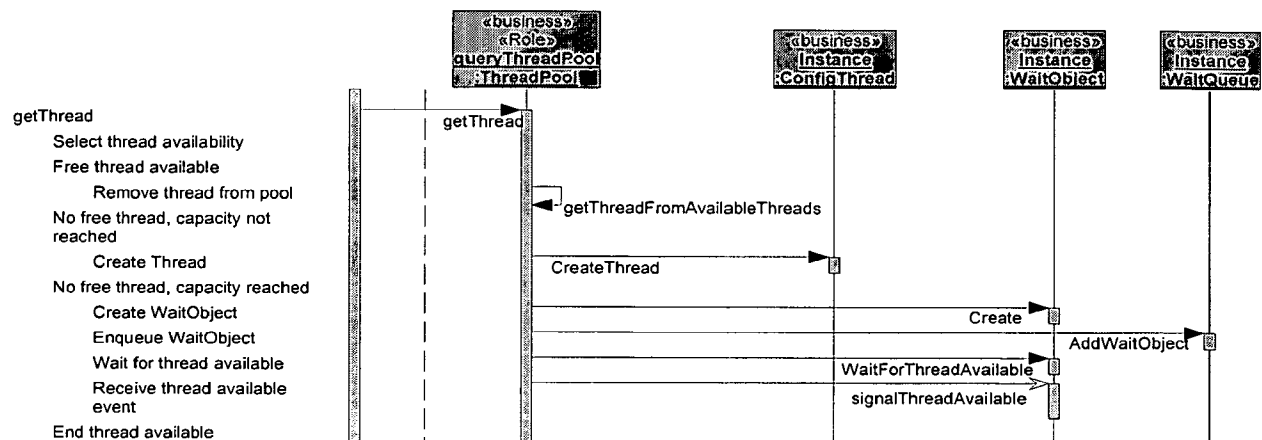


Figure 10 Get a thread from the pool

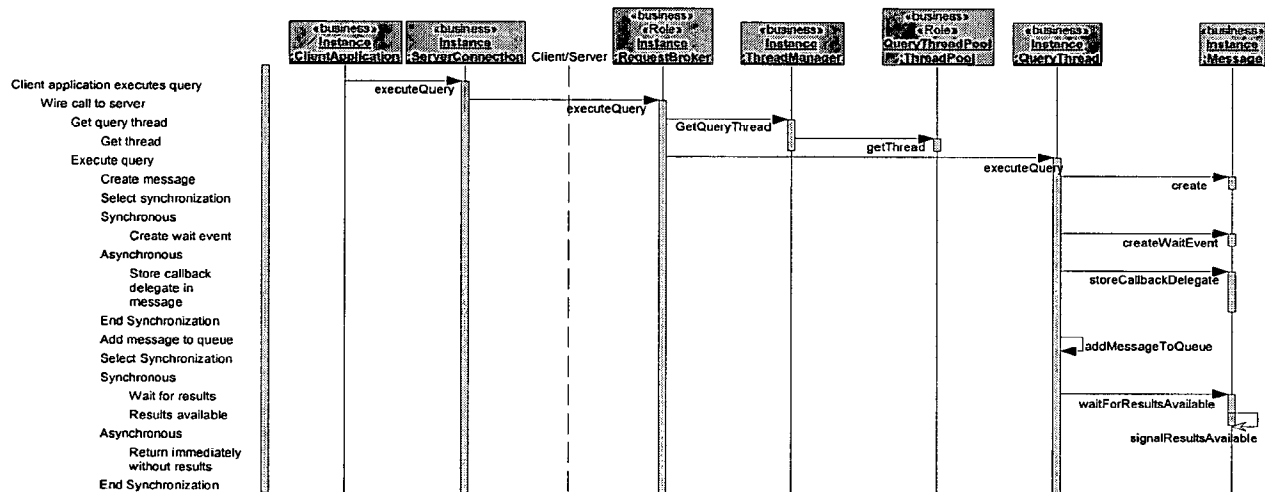


Figure 11 Prepare, Post/Send message

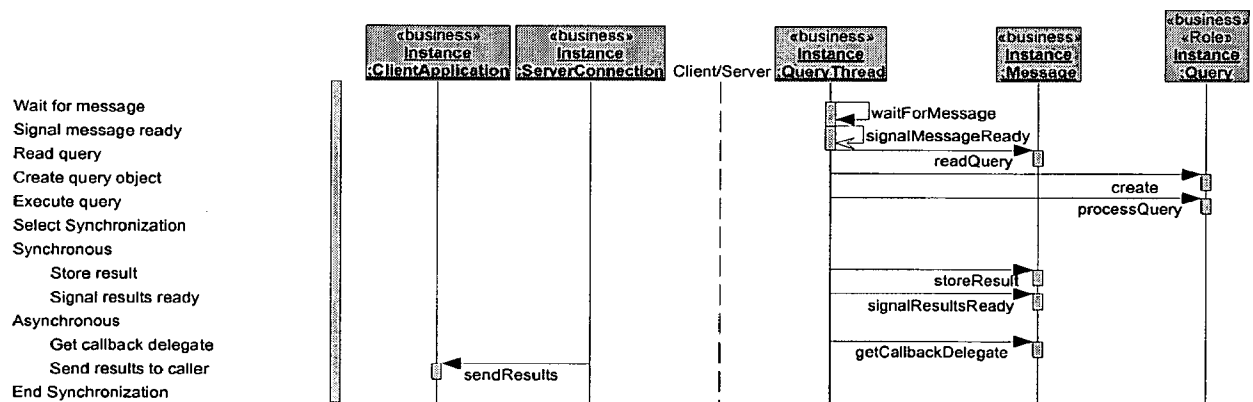


Figure 12 Process a message on the database thread

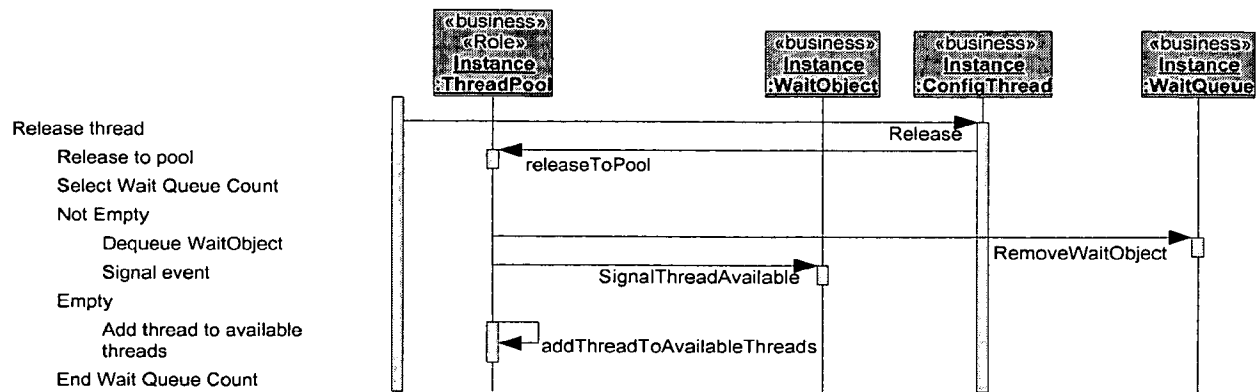


Figure 13 Release database thread

4.4 Query Framework

4.4.1 Description

The biggest change for clients is the introduction of the Query Framework. A Query is written in C# and coded against the XM layer which is a thin veneer C++ layer interface between the managed world of .NET and the unmanaged XM classes of HawkDB.

The Query semantics allow the user to specify whether the request is to register with the database for updates.

These query objects are very dynamic. A client asks for a query to be executed and gets the result as XML. The client request identifies the .net assembly and query. The framework loads the assembly if necessary and executes the query. This provides good decoupling of development paths.

Direct services like import/export, navigation, download/upload could be implemented in this framework too.

We store xml chunks in the cache. These chunks are loosely coupled by nature. The query objects created to load and save these xml chunks do nothing more. There's no business logic here. Now we produce/cache exactly the kind of data we want to transport around the system, ie XML. It's a flexible architecture that's extendable and can be molded by the client application writer to suit their needs.

The Query engine is shown below.

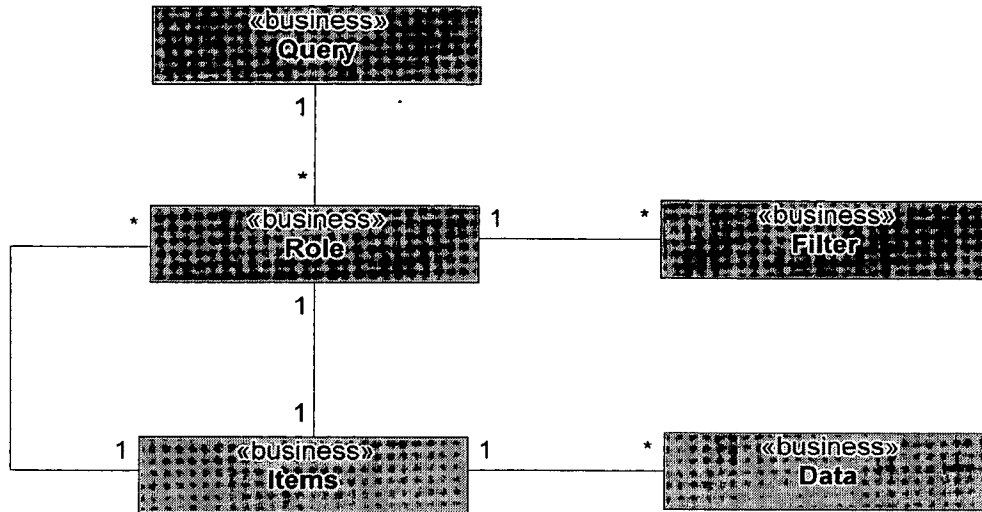


Figure 14. Query Model

4.4.2 Updating Queries

To improve the efficiency of saving changes to the database the query syntax could be extended to include property overrides. Thus executing the query would set object properties instead of reading them. This would be an alternative to submitting an update command script which would need to be compiled and executed.

4.5 Command Framework

Commands are submitted to the server in the form of JScripts. Each command is compiled and executed. The scripts are coded against the Server Configuration Model. At this time the exact syntax of the scripts has not been defined.

4.6 Service Framework

At this time there is specific service framework. The existing Client/Server model is extended to add service capability.

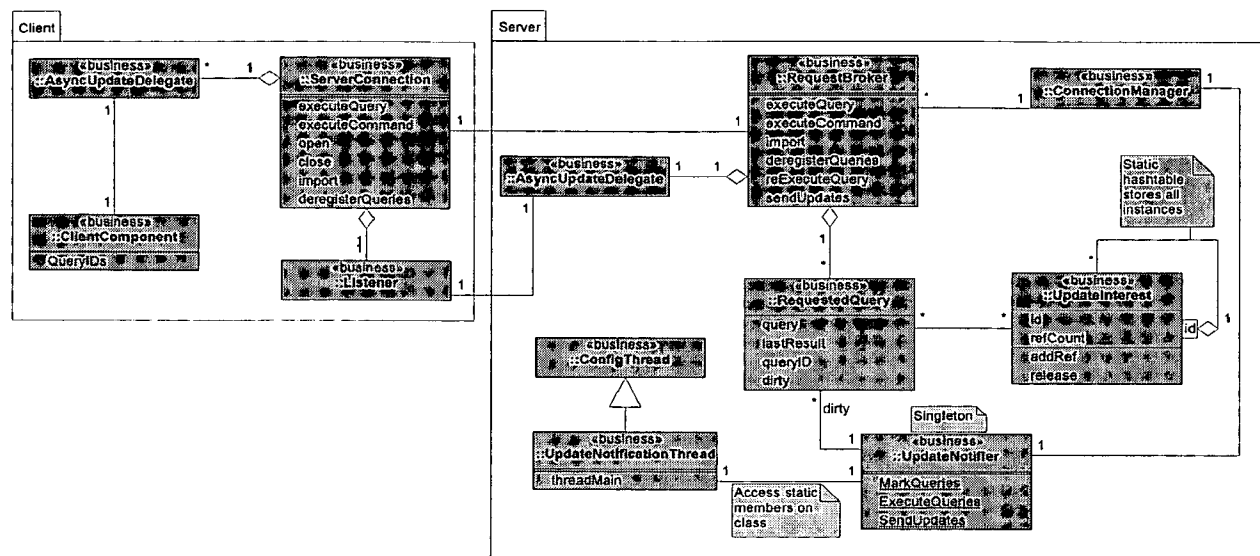
4.6.1 Import

ServerConnection and RequestBroker are extended to include an import method that takes a managed stream and an ImportCallback delegate.

In the server CommandThread is extended to include an import method. A new class ImportMessage derives from Message and is used to communicate the stream and import callback to the database thread. The database thread calls import on the database connection passing in the stream and import callback.

Inside the database connection the managed stream is wrapped in an unmanaged class that implements IStream. A pointer to this interface is passed into the database code where the import actually occurs.

4.8.1 Object Model



The query syntax indicates update registrations on roles and items. The query below requests updates when the PlantAreas role is modified and when the properties of modules are changed

As the query processor executes the query it will register roles and items for update notification as it encounters them.

It is possible to register with the same database object multiple times, either by the same client or different clients. We may make a database registration once. Thereafter the same registration ID will be shared. The class `UpdateInterest` encapsulates the registration ID and a reference count.

At the end of processing a query a list of `UpdateInterests` is returned to the `RequestBroker`. If this list is not empty we may create a `RequestedQuery`. This object stores the query and the latest version of the result. All the returned `UpdateInterests` are associated with the `RequestedQuery`. Each `RequestedQuery` has a unique ID that is returned to the client.

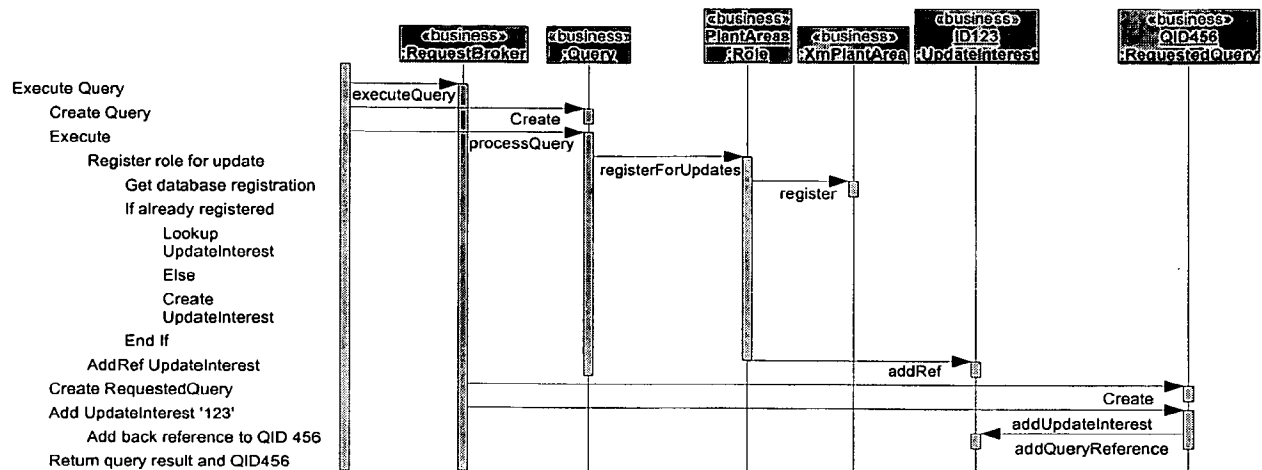


Figure 17 Execute a query with update registration

4.8.3 Notify Thread

The server runs a background thread to periodically process the list of changes. The processing is split into 3 phases.

The first phase gets the list of modified IDs from the database and marks all the related RequestedQuery objects dirty.

The second and third phases are deferred to the individual RequestBrokers. Initially the RequestBroker re-executes any queries marked dirty. By comparing the new results with the old we can determine whether the results have changed.

Once all the queries have been re-executed the results from all those that have changed are combined into a single XML document and sent back to the client via the AsyncUpdateDelegate.

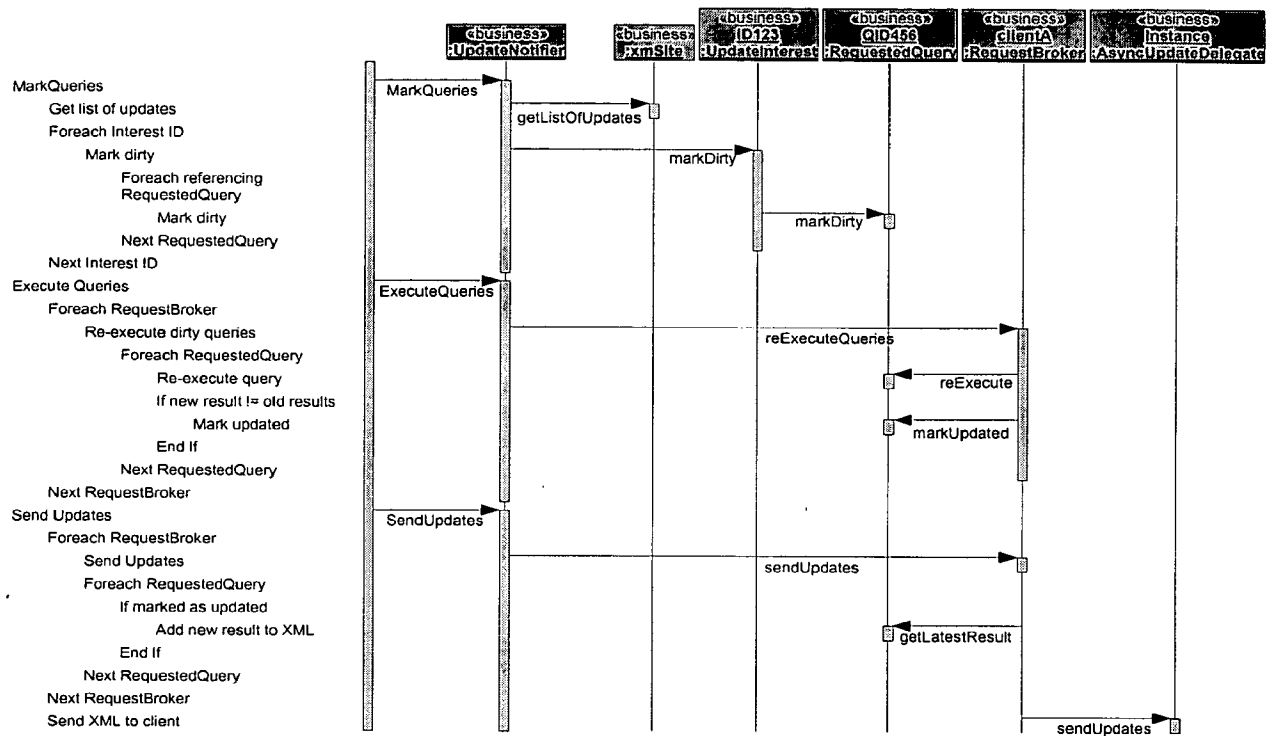


Figure 18 Notify Engine Execution

4.8.4 Execute a Command

After executing a command it is advantageous for the calling client to immediately receive any updates related to it. To achieve this behaviour a scaled down version of that which occurs in the notify thread is performed.

First the UpdateNotifier is asked to perform the mark phase of update notification. Then the second phase is performed for the calling application's RequestBroker. This re-executes all the registered queries.

Finally the results from the queries that have changed are combined into a single XML document and return to the client synchronously with the result of the command.

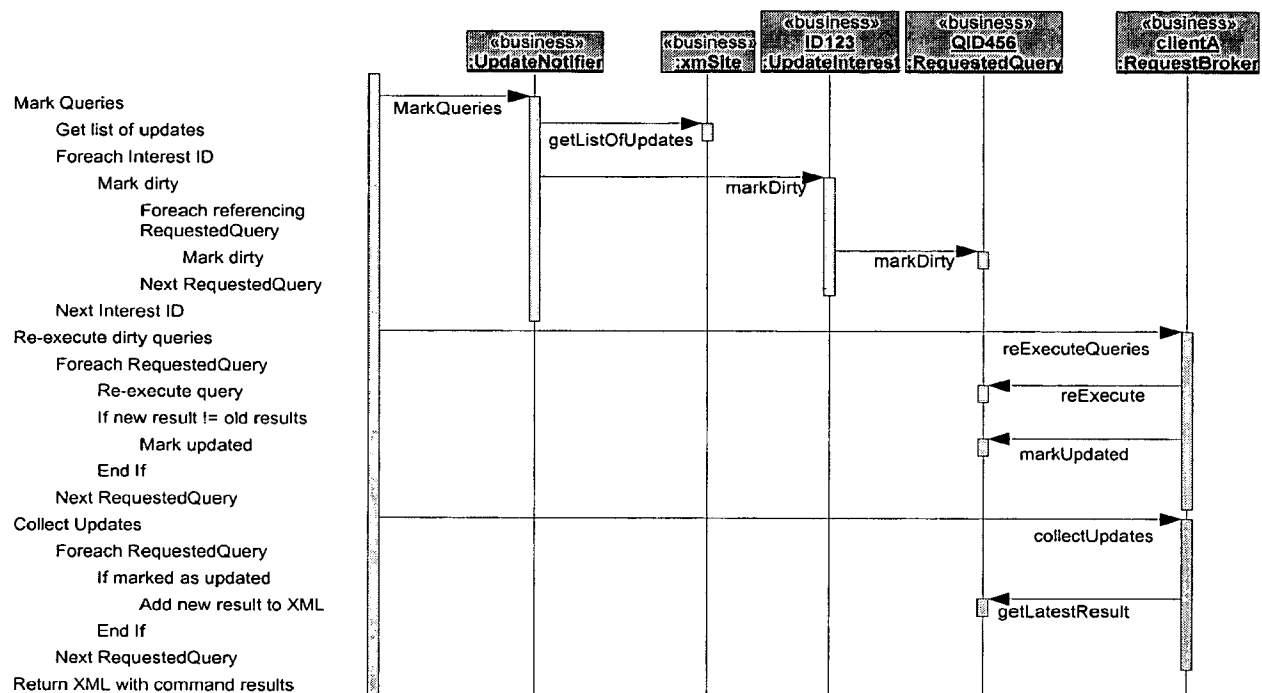


Figure 19 Process the updates for a single client after a command execution

4.8.5 Deregister a Query

In submitting a query to the server containing update notification request(s) the client will receive a query ID. This ID is used to inform the server when the client is no longer interested in receiving updates on that query.

When a server connection is shutdown the RequestBroker will forcibly deregister any queries the client may have forgotten to deregister.

In deregistering a query we call release on each of the UpdateInterests associated with the RequestedQuery. If the reference count on an UpdateInterest goes to zero the interest ID is deregistered with the database and the UpdateInterest is released.

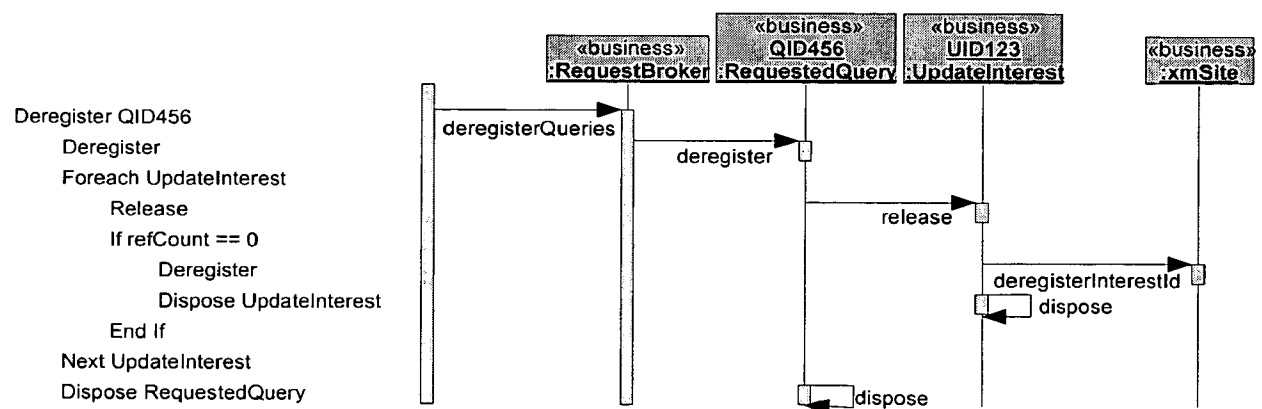


Figure 20 Deregister a query from update notification

Gas Blend Use Case Document

Index

1	Introduction.....	254
1.1	Overview.....	254
1.2	References.....	254
1.3	Intended audience.....	254
1.4	Terms and Definitions.....	254
1.4.1	Equipment Class.....	254
1.4.2	Material Class.....	254
1.4.3	Visualization Class.....	254
1.4.4	Process Library Item (PLI).....	254
1.4.5	Process Model View.....	254
1.4.6	Process Library.....	254
1.4.7	Process Module Subsystem.....	254
1.4.8	Process Module.....	255
1.4.9	Optimizer Function Block.....	255
1.4.10	Package.....	255
2	Configuration Workspace Overview.....	255
3	Use Cases.....	255
3.1	Create a GAS_BLEND package.....	255
3.2	User creates a Meter PLI Item.....	256
3.2.1	User creates a Process Equipment Class.....	256
3.2.2	User creates a Primitive Process Library Item.....	257
3.3	User creates a leg for the gas blend header.....	258
3.3.1	User create the Leg.....	258
3.3.2	User puts a pump PLI on the LEG.....	259
3.3.3	User puts a meter and valve PLIs on the LEG.....	260
3.3.4	User connects the smart items using a stream.....	261
3.3.5	Engineer exposes properties as public.....	262
3.3.6	Engineer generates default visualizations and control strategies.....	262
3.3.7	User customizes generated leg display.....	263
3.4	User Creates a GAS BLEND Header.....	264
3.4.1	User Creates PLI HEADER1.....	264
3.4.2	User added instances of LEG1 to HEADER1.....	264
3.4.3	User connects lets together with extensible STREAM.....	265
3.4.4	User exposes connectors in header.....	266
3.4.5	User save and generates control strategy and displays.....	266
3.5	The user creates GAS BLEND Process Module Class.....	266
3.5.1	User creates a Process Module class.....	267
3.5.2	User adds PLI instances to Process Module class.....	267
3.5.3	User adds PLI instances placeholders.....	267
3.5.4	User adds Gas Blend optimizer blocks.....	267
3.5.5	User save and generates.....	268
3.6	Engineer Create an Instance of GS_BLEND.....	269
3.7	Engineer downloads the GAS_BLEND instance.....	270
3.8	Engineer debugs an instance of GAS_BLEND.....	271
3.9	Engineer exports the GS_BLEND package from DV System.....	272
3.10	Engineer Imports the GS_BLEND package in a DV System.....	272

1 Introduction

1.1 Overview

This use case document is an attempt to describe at a high level the use cases to configure a Gas Blend package in a DeltaV system within the System architecture.

1.2 References

- System Configuration Workspace Framework document: An overview of generic configuration workspace.
- Workspace Framework: An overview of a generic workspace.
- System Technical Architectural: Discusses the System initiative and architecture.

1.3 Terms and Definitions

1.3.1 Equipment Class

Defines the structure and behavior of something physical like a tank or a pump.

1.3.2 Material Class

Defines the properties and behaviors of a type material, e.g. Gasoline.

1.3.3 Visualization Class

A class that defines the visual characteristics of a physical object such as a pump or valve

1.3.4 Process Library Item (PLI)

A smart object that is composed of an equipment class and optionally, visualizations, a single control strategy and a single material class.

Additionally, a process library item may contain instances of other process library items building up composite process library items

1.3.5 Process Model View

The view displayed when editing a process library item.

1.3.6 Process Library

A library in the DeltaV hierarchy, it contains PLI, visualization classes, equipment classes and material classes.

1.3.7 Process Module Subsystem

A subsystem that runs on a workstation or application station and it will have instances of optimizer function blocks.

1.3.8 Process Module

A module that runs within a process module subsystem and it provides a higher level of control or optimization than the controller. Generally, process modules have highly specialized blocks to optimize a portion of a process.

1.3.9 Optimizer Function Block

Special function blocks that control parts of a process that are put in a process module and run in the process module subsystem, which resides in a workstation. FRS, third party integrators or even customers can create optimizer Function Blocks.

1.3.10 Package

A package is a self-contained unit that encompasses a logical amount of work. Packages can be built up to create libraries of functionality. A DeltaV System can contain multiple packages.

2 Configuration Workspace Overview

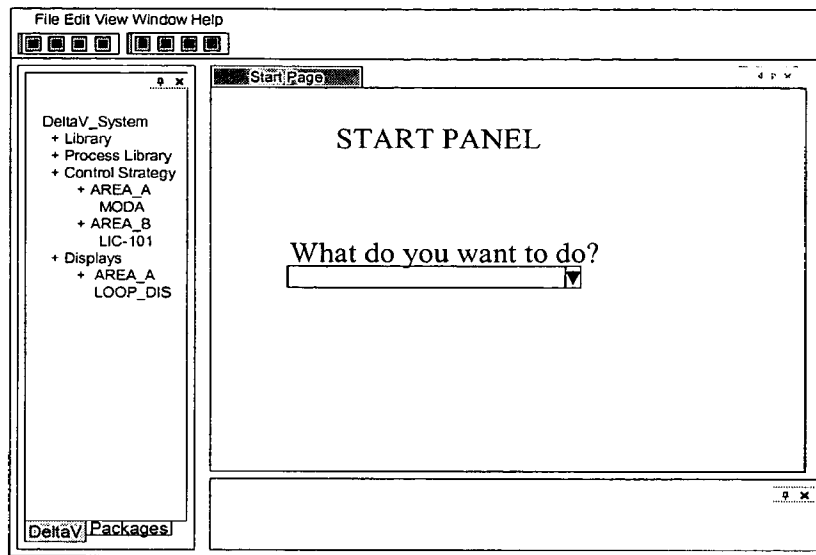
When a configuration engineer, user, starts to configure a system they will start up the configuration workspace. A startup the workspace will have a main panel that the majority of configuration will take place. This is where the user will configure graphics, process modules, control strategies and the physical network. There will be a sub panel that is by default docked to the left. This panel will contain several items that may be in a tabbed window. One of the items will be the main DeltaV tab and will contain a tree to explore the DeltaV hierarchy in a manner similar to the current DeltaV Explorer. Items in this can be drag-dropped to main panel for the user to configure. A second tab will be items that the user has created in the current package such as, control module classes, process models, graphics, etc. Another possible tab will be the pallet of primitive graphics that come pre-populated with a DeltaV system as well as composite graphics that have been configured in the system. This could be the pallet and an explorer or browse view that the user can use for drag drop operations and side configuration. The packages created for this system will also be displayed under the packages tree item in the DeltaV Explorer tree.

New systems will ship with a default package, DeltaV. Pre-populated items that ship will belong in the default package. Items can't exist outside of a package. New items that a user creates can be added to the default package, but the preference would be to create a new package maintaining a separation between default items and customer created items. A user or system integrator can create 1 to n packages. A package is a self-contained unit that encompasses a logical amount of work. Packages can be exported and imported as a unit into other DeltaV systems thereby adding value by building up a library of packages that can be reused.

3 Use Cases

3.1 Create a GAS_BLEND package

User will start up the configuration workspace application. A tree will appear in a panel docked to the left or the right. The configuration workspace may appear something like the following picture.



User will select new package and enter “GAS BLEND”.

Results: A package named GAS BLEND will be created under the packages library item. The package will be empty except for the necessary pre-populated portions of the package.

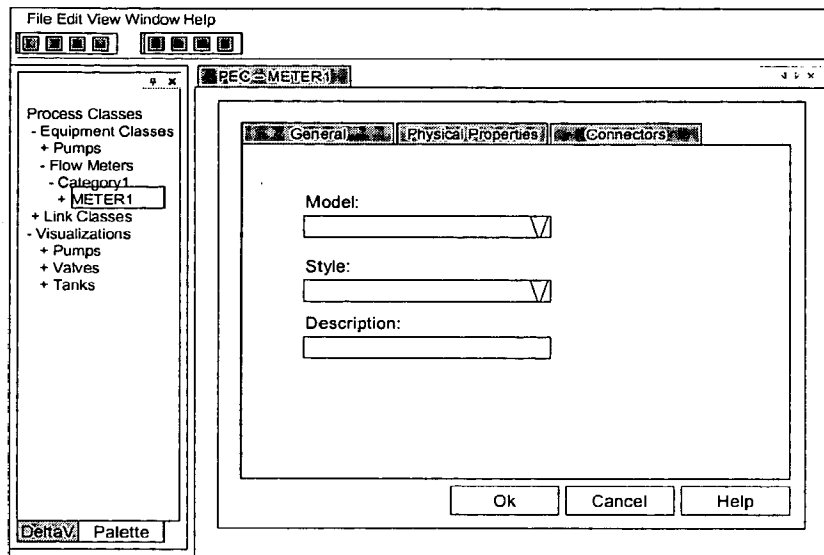
3.2 User creates a Meter PLI Item.

In order to create the gas blend package the user determines that there is not an adequate meter and one must be created.

3.2.1 User creates a Process Equipment Class

First the user creates a new Process Equipment Class of type meter and enters the name METER1. The main panel displays the properties pages of the meter and the user enters characteristics of this particular equipment and selects OK.

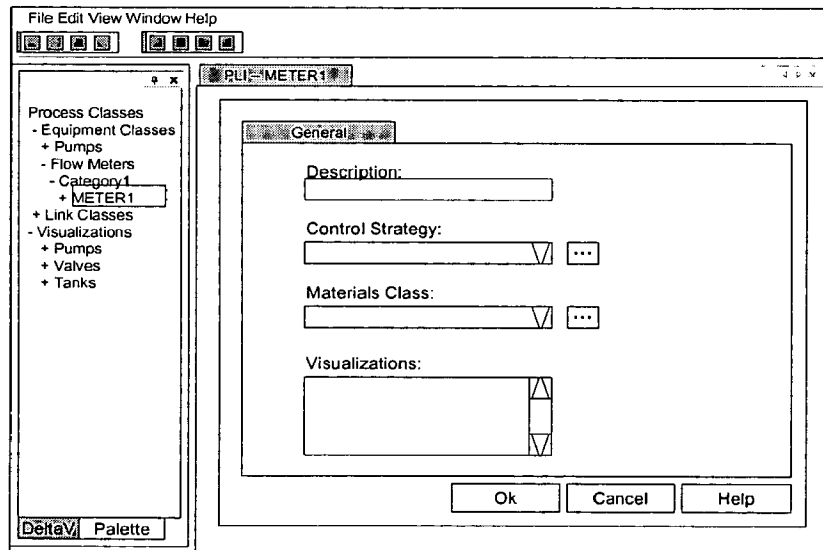
Process Equipment Class



3.2.2 User creates a Primitive Process Library Item

The user then creates a process library item for the new equipment class and enters the name METER1. The user selects properties and the main display will show the PLI properties pages. The PLI item is associated to the Process Equipment Class created above by drag-n-drop or by using the property pages. The user then associates an existing or new visualization or customizes an existing visualization.

Process Library Item



The user decides that they will use an existing visualization and selects one from the Display class palette. If the user wants to customize this display in any way they create a new display class².

Finally the user associates a control strategy to the meter. The user decides that it is not necessary to create an entire module for the meter and that a PID block is adequate for control. Again, the user can do this by drag-n-drop or from the property page. If the user had decided to create a control module class they would have the option to open a new window in the main panel to edit the algorithm.

The user wants to change some default values on the PID block to best represent the characteristics of the meter. The user selects Control strategy and the PID block is displayed as a new tab on the control strategy view, which displays as a new tab on the main panel. The user can override parameter values, create extensible parameters, and other actions that can generally be performed on a primitive function block.

The user selects the “Save” command and the PLI and control strategy changes are persisted in the database.

3.3 User creates a leg for the gas blend header

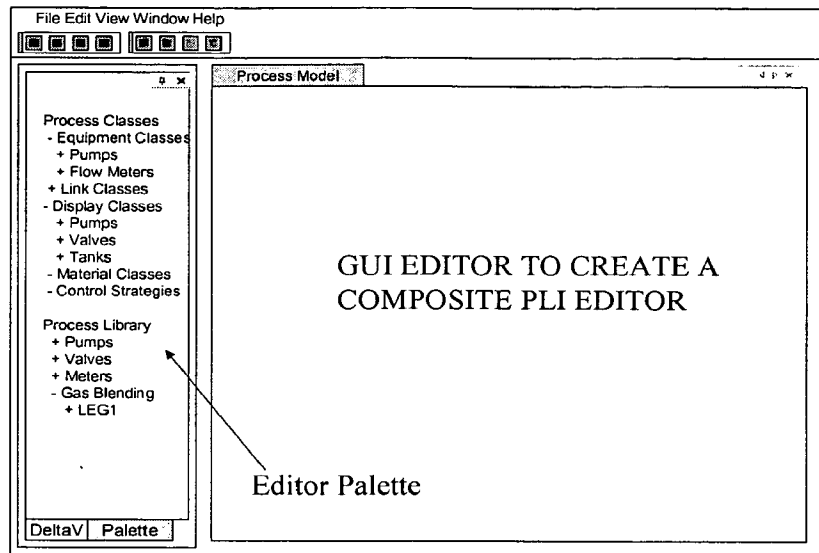
The user will create a composite Process Library Item that will include several basic process library items. In this case the user is going to create a leg, which contains a pump flow meter and valve and saved as a new Process Library Item under the DeltaV hierarchy and associated with the package.

3.3.1 User create the Leg

User selects new Process Model and then enters a name of LEG1. An empty process model view is displayed within the main panel of the configuration workspace. By default a process model palette window will be displayed and will be rooted at the process library items in the DeltaV hierarchy. The palette window will be docked under the explorer tree as a separate tab but can be moved to be a separate docked location.

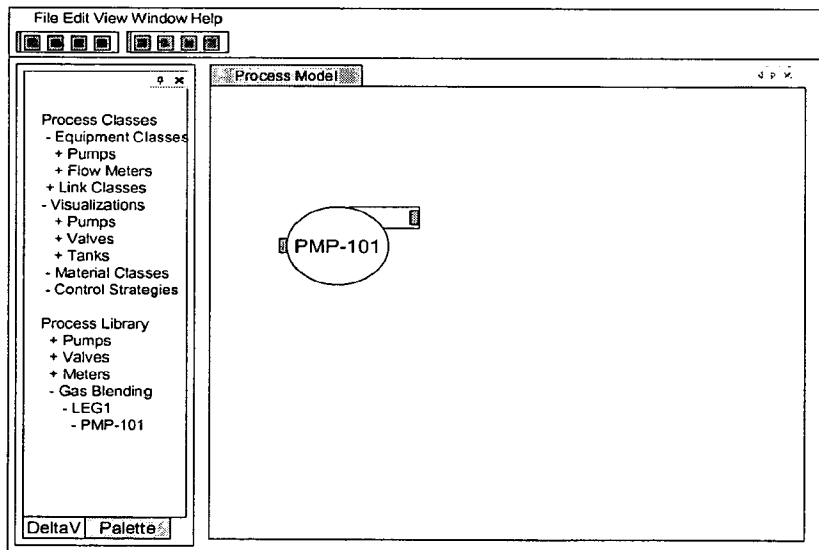
The result could look something like below

² The user may create a new display class or may override things in a display class.



3.3.2 User puts a pump PLI on the LEG

The user will navigate the existing PLI items in the palette window and under the pump category, and maybe several subcategories, to select the correct pump identified on the PID drawing. The user drags the pump from the palette to the process model and gives it a name, generally the name that is on the PID drawing.



At this point the user has the option to take the default visualization or select a different visualization. The user can select an alternate visualization that is associated with the pumps PLI definition or the user can enter select a visualization from the visualization classes that is compatible with the pump's PLI definitions Process

Equipment class. This can be done multiple ways such as drag-n-drop or browsing, or editing the pump instance properties.

- Visualizations
 - Pumps
 - External
 - PV_A
 - PV_B
 - PV_C
 - PV_D
 - Internal
 - Electrical
 - + Valves
 - + Tanks
- Material Classes
- Control Strategies

When this occurs an additional visualization³ is created for that instance of the pump on LEG1⁴.

The user then has a chance to override the Control Class for pump PMP-1. In predefined PLI items a control class may or may not be defined. For simple items the control class may be a primitive block such as a DC block for a pump. The user will have the chance to override the control class⁵ and enter another existing block, or module class as well as create a new composite block or module class. The user mechanism for this will be the same as the visualization. However, one control strategy can be associated with a PLI. If the user changes the control strategy the change affects the instance of the pump not the pumps definition⁶.

At this time the user decides that the existing control class is sufficient

3.3.3 User puts a meter and valve PLIs on the LEG

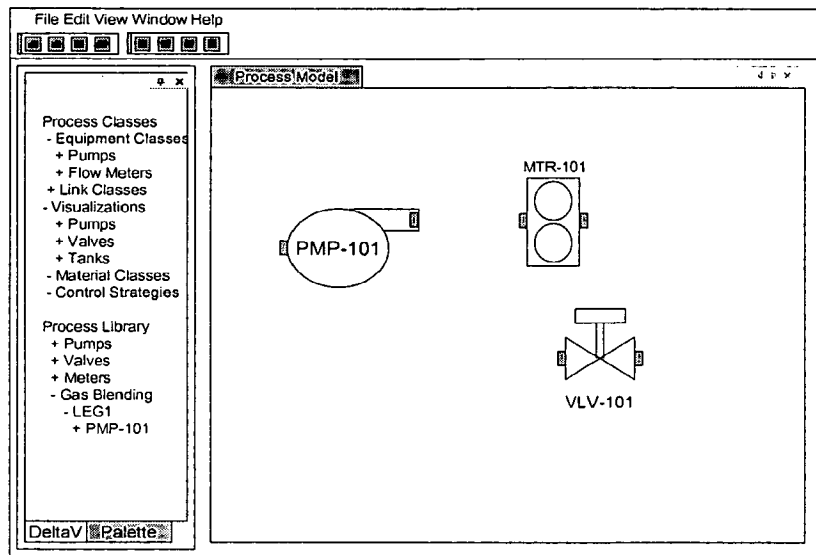
Now the user will put a meter, MTR-101, and valve, VLV-101, on the leg. The user follows the process above for creating a pump and to create a meter by choosing the meter created above and chooses a preexisting value from the library.

³ Process Library Items can associate to multiple visualizations and one control strategy

⁴ Assumption is the pump on LEG1 behaves similar to definitions and instances. An alternative would be to modify the definition or create a new definition for the pump PLI.

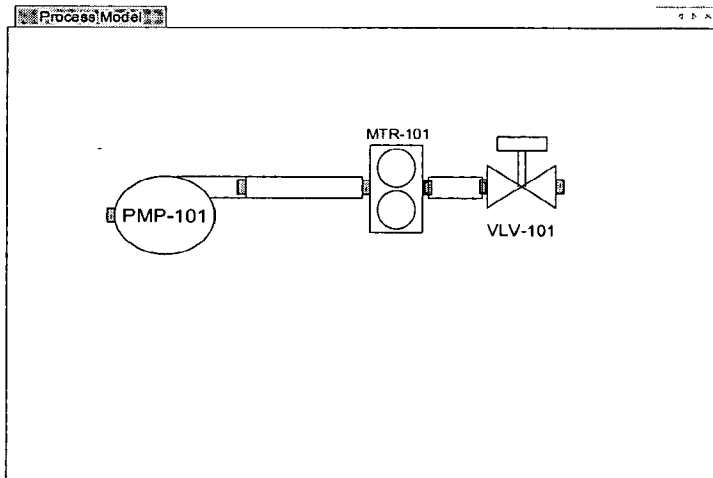
⁵ Unlike visualizations, a PLI item definition and instance can be associated with one control strategy.

⁶ Assumption is PLI items work like definition and usages.

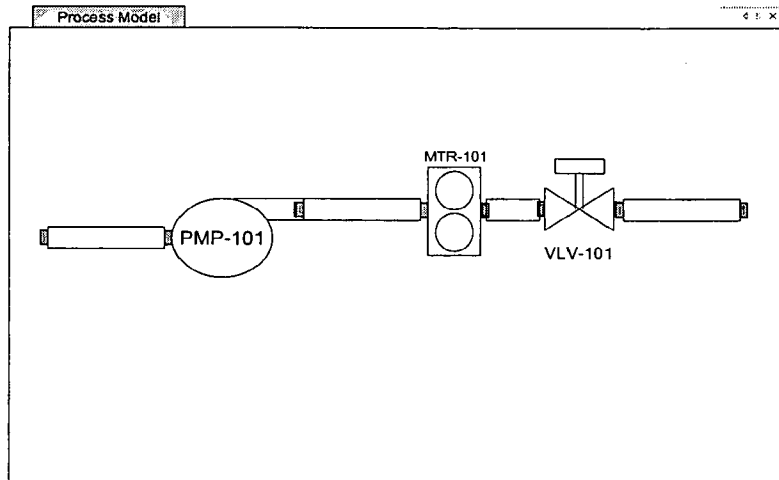


3.3.4 User connects the smart items using a stream.

At this time the user needs to connect the smart items (PLI) on the process model. The user navigates to the palette with the streams, which are a specialized PLI types, and selects an 8" stainless steel pipe with XYZ characteristics. The user then selects the out connector from the pump and drags the mouse to the in connector of the meter. The result is an instance of the pipe stream is created which connects the pump and meter.



Also, the user wants to expose an input and output to the leg to be used on the header. To accomplish this the user will create another stream but connect the output connector to the input connector of the pump. Then the user will select the input connector on the stream and change its properties to public. The process will be repeated for the output but the stream will be connected to the pipe.



3.3.5 Engineer exposes properties as public

The configuration engineer wants to expose additional properties of the leg as public. The configuration engineer selects the meter and the properties of the meter are displayed in the property window. (This is another window that will be docked or tabbed within a docked window). The user selects the PV of the meter and selects new external interface with a name of FLOW_RATE_PV. The user selects the SP and creates a new external interface FLOW_RATE_SP. Additionally, the user wants to expose the value position as a read-only public interface this done the same way except the user designates this public property as read-only.

At this point the user selects save command, which persists the process model changes into the database.

3.3.6 Engineer generates default visualizations and control strategies

Now the user wants to group the pump value and meter together. This is given a loop tag of FCC-101. The user then selects the Generate Command. The generate step will create a default control strategy. In this case a Control Module Class will be generated with a name of FCC-101. If any of the smart object's (PLI) control strategy had been a module the results would be an Equipment Module Class. In this case the Control Module Class will use the designated control strategy blocks defined for the pump (DC), valve (PID) and meter (AO) to generate the control module.

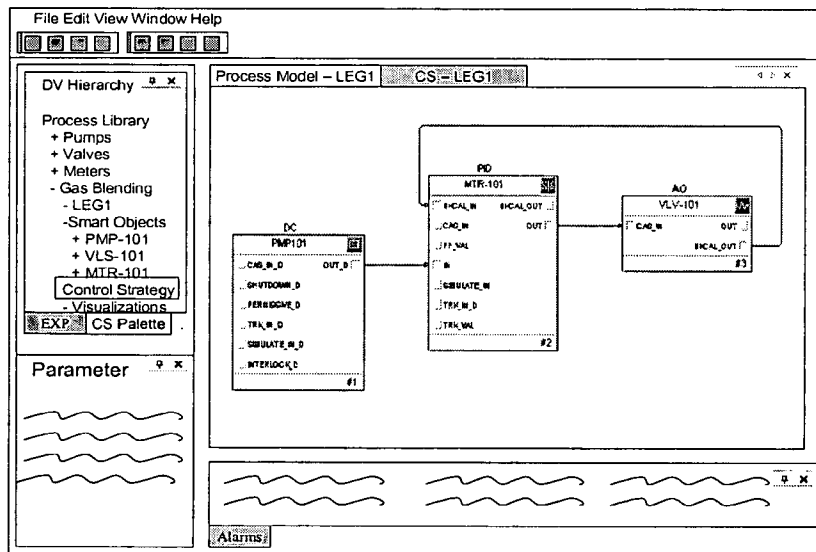
In addition, the generate command will create visualizations for LEG1⁷. There may be one or more visualizations generated. This will be based upon the visualizations that are associated with the PLI instances on LEG1. Some examples are detail operator display class and overview displays class, and faceplate display class.

The generated module class and display classes will be associated. The operator display will have links to the control strategy, faceplates and other objects, which will have links back links to the original objects.

At this point the Hierarchy view displays the control strategy and displays under LEG1 in appropriate categories and additional menus and toolbars are enabled. The configuration engineer can then select to edit the generated control strategy. In the main display a new tab appears and the Control Module Class appears. The Process Model is hidden but easily accessible as another tab on the main display.

The user then can edit the control strategy as necessary⁸. The configuration engineer will then be able to save the Control Strategy (Control Module Class).

The Control Module class will appear in the GAS_BLEND package and the overall DeltaV hierarchy under the Advanced Definition / Control Class /Module Classes and under LEG1.



3.3.7 User customizes generated leg display

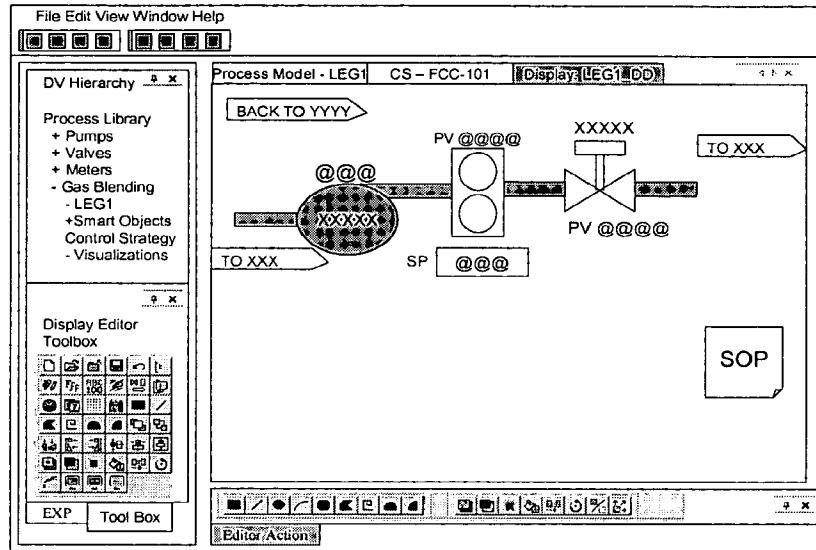
The configuration engineer wants to modify the generated display for the leg. The engineer selects to edit the display. The main panel in the configuration workspace displays a new tab and the previously generated Operator Display is shown. A display editor palette is shown which is the toolbox of graphic items for the display

⁷ In some cases the such as faceplates there may be nothing to generate just the tag needs associated to the correct faceplate template

⁸ The generated control strategy and displays may be loosely coupled. If the user changes the process model the control strategy is not updated until a regenerate is performed.

editor. This is either as separate docked window or as a tab within the explorer window⁹.

The engineer decides that some of the generated graphics need adjustment. The user selects the detail display graphic and moves it to the desired location and additionally changes some of the properties of the selected graphic. Then the user adds a document visualization, which is then linked to a foreign document that contains the operation procedures for this leg.



The user selects the save command and the display is persisted in the database.

The user then closes all items in the main panel, control strategies, displays and process model views, etc.

3.4 User Creates a GAS BLEND Header

The user wishes to create a process library item (PLI) to represent the header of the gas blending. The header will be made up of 8 instances of LEG1 created earlier.

3.4.1 User Creates PLI HEADER1

User selects new Process Model and then enters a name of HEADER1. An empty process model view is displayed within the main panel of the configuration workspace. By default a process model palette window will be displayed and will be rooted at the process library item in the DeltaV hierarchy.

3.4.2 User added instances of LEG1 to HEADER1

The engineer navigates to the PLI item LEG1 that was created earlier. The user then drag-n-drops an instance on the diagram and for now accepts the default instance name LEG1_1. The leg is rendered as a function block, unless the user has assigned a

⁹ Note: All docked windows will have the ability to have multiple tabs.

graphic for the leg. There will be an input and output connector which represent the exposed portions of the stream. Additionally, when the LEG1 instance is selected the property window will display the public properties LEG1 such as FLOW_RATE_CV, FLOW_RATE_PV and FC-101/SP.

The engineer within the process model view selects the leg and copy/pastes 7 additional instances named LEG1_2 to LEG1_8. The engineer then using in-place edit renames the legs to the appropriate names for the GAS BLEND header¹⁰.

The new names of the legs are:

- REFORMAT
- THERMALLY_CRACKED
- CATALYTICAL_CRACK
- POLYMERIZED
- LSR
- NBUTAIN
- ALKYL

3.4.3 User connects lets together with extensible STREAM

The engineer then ties these together with an extensible stream. The engineer using a drag-n-drop, typing or a wizard creates a stream with a default name STREAM1. The engineer selects the stream and indicates that this stream has 8 inputs. The stream is redrawn with 8 inputs and one output connector. The user then rearranges the stream to create waypoints to allow the stream to be represented as a series of horizontal inputs and a vertical section and a horizontal output. At this point the user attaches the legs to the stream inputs¹¹.

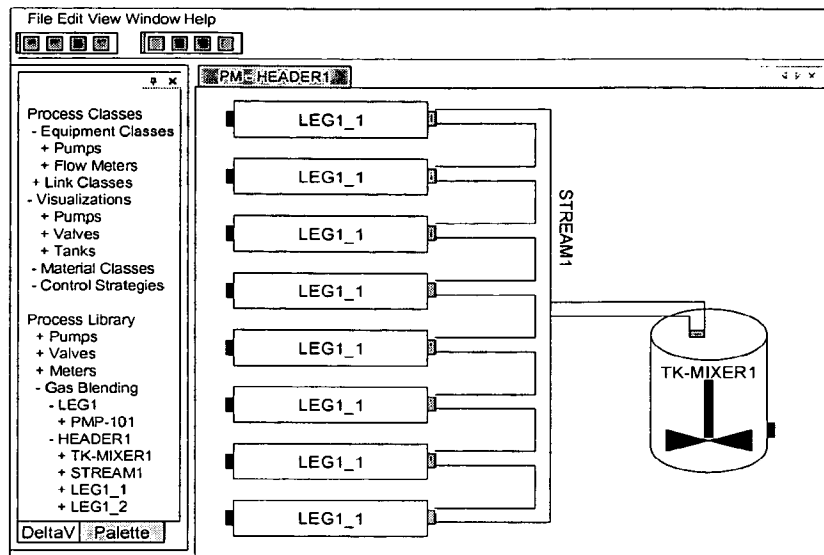
Then the engineer creates a mixer tank for the header. At this point the user decides to navigate the Process Model view's palette and select the appropriate mixing tank from the mixing category. An instance of the PLI item is created with a default name of TK-101. The user decides to rename this to TK-MIXER1. The output of the STREAM1 is attached to the input of the mixer tank TK-MIXER1. The mixer tank is a simple tank and has a control strategy associated of a DC block since a paddle that is controlled by an integrated motor does the mixing.

The result may look something like the figure below¹².

¹⁰ May be IN1 to IN8 and renamed at when the feed tanks are attached.

¹¹ Possible that attaching multiple inputs to a stream automatically creates additional inputs on the stream

¹² Leg instance will show as a function block if there is no default visualization. Most likely the visualization represented when creating other PLI item may be the overview display generated.



3.4.4 User exposes connectors in header

The user exposes the public connectors and public properties of HEADER1. The user multi-selects the public inputs on the 8 LEG1 instances as well as the output connector on the TK-MIXER1 and designates those as public. The public connectors may be shown in a different color such as blue. Users of the HEADER1 will see connectors named REFORMAT_IN, NBUTAIN_IN etc.

3.4.5 User save and generates control strategy and displays

The user then saves the process model view. The user then generates a default control strategy. The control strategy generated is an Equipment Module Class (EMC) named HEADER1_EM since the definition of LEG1's control strategy is already a module class.

The user then requests to generate a display class for the HEADER1. The display class is named HEADER1. By default operator display classes that are generated will have links to other operator display classes for each public input and output connector. For example, the HEADER_DISP will have links to the LEG_DISP display class. When the operator is drilled into the LEG there will be a link back to the correct header instance.

3.5 The user creates GAS BLEND Process Module Class.

The user has now created several building blocks for the gas blend system. These are all class-based items in the process library. The user is now ready to create a Process Module using the items created before and the optimizer's blocks for blending gas. A process module will run within a workstation not the controller.

3.5.1 User creates a Process Module class

The user navigates to the Process Module area of the DeltaV Hierarchy or the GAS_BLEND package hierarchy. The user then creates a new Process Module with the name of GAS_BLEND_PM and with an algorithm of “optimizer”.

3.5.2 User adds PLI instances to Process Module class

The user chooses to add an instance of the PLI item HEADER1. The header one instance appears on the process model view as a block with the 8 inputs and 1 output displayed¹³. The user renames HEADER1_1 to GB_HEADER. The user then creates a feed tank. The user selects a feed tank from an existing tank category in PLI section of the library. The feed tank by default is named TK-101. The user creates a level indicator, LI1 for the tank and exposes the capacity property for the tank as public¹⁴. Finally, the user then creates a stream by selecting a stream PLI and attaches it to the output of the TK-101 and to the REFORMAT_IN on the GB_HEADER. This tank is copied for the other 7 feed tanks, TK-102 to TK-108 and attached to the header.

The user then creates blend tank by dropping an existing tank PLI and names it TK_BLND. The user then creates another stream and attaches the stream to the output of the header, GB_HEADER to the input of the TK-BLND.

3.5.3 User adds PLI instances placeholders

At this point the user needs to offer some customization to the GAS_BLEND process model. The configuration engineer realizes that some implementations a booster pump will be needed one or more of the feed tanks TK-101 to TK-108. The user creates a pump and drops it on the stream. Using some special control key such as the control shift the user is able to automatically insert that into the existing stream and a new stream is create and the in and out connectors of the stream and pump are automatically connected. The user renames the booster pump PMP-101 to PMP-108 (not shown). The user marks these as placeholder item, that is a piece of equipment, which may might not exist in a customer’s plant.

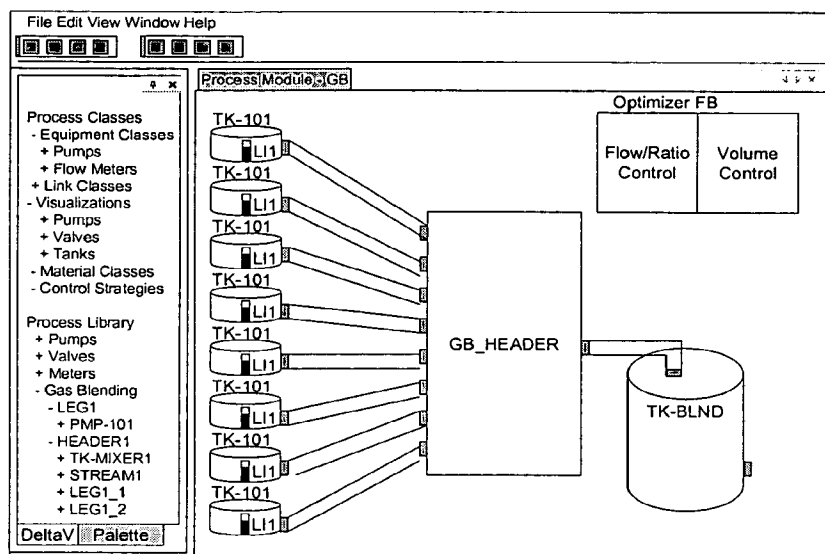
3.5.4 User adds Gas Blend optimizer blocks

Now the user needs to create instances of two gas blend optimizers for the gas blend algorithm¹⁶. The user navigated to the optimizer’s category within the DeltaV hierarchy. The user expands the “GAS_BLEND_OPT” optimizer category and drags an instance of the Flow/Ratio Control optimizer to the Process Model. The user then drags Volume Control optimizer. These are represented as function blocks on the process module and are named FRC1 and VC1 respectively.

¹³ Note: the representation of the PLI item can be changed from a block to anything by associating the PLI items to a visualization class

¹⁴ The user would probably create a composite PLI with the tank and level control and create 8 instances on the GAS_BLEND process module.

¹⁶ It is believed that the optimizer function blocks are somewhat similar to primitive function blocks. They are not created and configured via the DeltaV configuration workspace but are standalone plug-ins that run within a Workstation’s Process Model Subsystem. There is a loosely coupled representation of them in the database for configuration of process modules.



The user then associates the FRC1 to the input of the GB_HEADER and the to the mixer within the GB_HEADER¹⁷. The user then associates the volume control with the blend tank and the FRC1 optimizer block.

The user then associates the optimizers with the input tanks level indicators LIC-101 to LIC-108, the instances FLOW_RATE_PV and FLOW_RATE_SP outputs of the 8 legs and the blend tank level PV. This is done so the optimizer can determine if there will be sufficient feed stock for the blend operation as well as if flow of a leg is unable to reach the set point, pacing would be required. If insufficient feed is unavailable the optimizer may be able to use an alternate combination of feed, which may not be most cost effective but still result in a product that is in specification.

3.5.5 User save and generates

The user saves this to the configuration database and the GAS_BLEND process module is updated.

The user then selects to generate the control strategy and operator displays. An equipment module class GS_BLEND_EM is created and will control the GS_HEADER_EM equipment module class as well as control the interaction with the optimizer function blocks. The operator display class is created named GS_BLEND_DISP and it will contain graphics of the feed tanks, header and blend tanks as well as a graphical representation of the optimizer blocks, which run on a workstation. The operator display that is generated will by default have links to the GS_HEADER display class, the Equipment Module GS_BLEND_EM class. Additionally, the graphics such as the tanks and even the header will have the links to the faceplates, which will be class/template based and data driven therefore will not require any user configuration. These will be saved in the appropriate location in the DeltaV and GAS_BLEND package hierarchy.

¹⁷ This requires certain parts of the mixer to be public within the header.

3.6 Engineer Create an Instance of GS_BLEND

The engineer would like to test the GS_BLEND package. To do this an instance of the gas blend package may be created and assigned to a controller (or workstation if simulating).

The user then creates an instance of the GAS_BLEND process module by navigating to AREA_A and selecting new process module instance. The user is able to browse to the process module classes and chooses the GAS_BLEND process module class. At this time the system generates many module and display instances at that time. For the GAS_BLEND model the following hierarchy will be created.

Instances	Definition	Type
Control Strategies		
AREA_A		
GAS_BLEND_1	GAS_BLEND	Process
Module		
GAS_BLEND_EM_1	GAS_BLEND_EM	Equip
Module		
GS_HEADER_1	HEADER1	Equip
Module		
REFORMATE_1	LEG1	Control
Module		
THERMALLY_CRACKED_1	LEG1	Control
Module		
CATALYTICAL_CRACK_1	LEG1	Control
Module		
POLYMERIZED_1	LEG1	Control
Module		
LSR_1	LEG1	Control
Module		
NBUTAIN_1	LEG1	Control
Module		
ALKYL_1	LEG1	Control
Module		
Displays		
AREA_A		
GAS_BLEND_DISP_1	GAS_BLEND_DISP	Display
GS_HEADER_DISP_1	HEADER1	Display
REFORMATE_1	LEG1	Display
THERMALLY_CRACKED_1	LEG1	Display
CATALYTICAL_CRACK_1	LEG1	Display
POLYMERIZED_1	LEG1	Display
LSR_1	LEG1	Display
NBUTAIN_1	LEG1	Display
ALKYL_1	LEG1	Display

In some extremely complex scenarios, it may take a few minutes to generate the instances. While the instances are being generated the user will not be blocked and have to opportunity to perform other work such as commission a controller, enable card channels etc.

After the instances are created the main panel will display a table view of the gas blend instance with the public and exposed properties visible. The engineer will then be able to override the public properties such as I/O assignment, gains, alarm limits with in-place edits.

The engineer will also be able to display the properties of the GAS_BLEND1 instance or subordinate instances by selecting the property command. This creates a new tab within the main display and is a modeless dialog or displayed as a specialized docked window on the configuration workspace. At a minimum this should not be modal to all other to select other objects in the hierarchy and see their properties without the multiple steps to cancel the current property dialog and bring up a new property dialog. Some examples of the properties that might be overridden the size of the stream, pump capacity for the legs, if booster pumps exist or not.

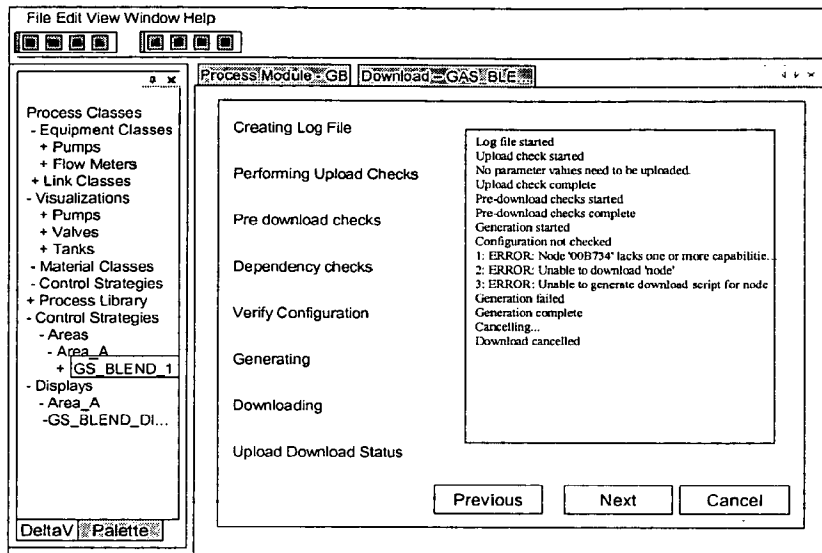
The user then assigns the process module to a workstation that has the process module subsystem enabled, valid nodes are the ProPlus w/s or an Application station. Additionally, the user assigns the modules to a controller, or w/s if in simulate mode).

3.7 Engineer downloads the GAS_BLEND instance

If the engineer decides that they need to download the entire system since nothing has been downloaded then the engineer can simply download the Physical network. This will download the process module, equipment and control module instances and operator displays.

If the engineer decides that they would like to download the GAS_BLEND module and subordinate modules as well as operator displays the user will select the GAS_BLEND_1 instance in the Control Strategy/AREA_A and choose download additionally, the user will select to download the GAS_BLEND_DISP under the Displays/AREA_A location in the DeltaV hierarchy.

When downloading is initiated a wizard will appear as a new tab within in the main panel. This will be a modeless window and the user can navigate to other windows without dismissing the download dialog. The user will navigate the wizard and initiate the download. After all the questions have been answered, and the download is generating or progressing, the user will be able to select another tab in the main panel or open up other items from the main DeltaV tree. The user will not be blocked at the UI while the download is progressing.



3.8 Engineer debugs an instance of GAS_BLEND

The engineer wants to debug the instance of the GAS_BLEND process model that has been created and all associated equipment and control modules, operator displays and links within the operator displays to other displays, control strategies and faceplates, and advanced control systems.

The engineer has successfully downloaded the GAS_BLEND configuration. The user selects the GB_HEADER_1 equipment module instance within the gas blends system and selects debug option. The control strategy for the class appears in the main panel in a debug mode with values updating. From the GB_HEADER_1 module instance the engineer is able to navigate to the parent and/or child module instances with a single mouse click, navigating or browsing is not necessary.

The engineer would like to test the operator displays for the GAS_BLEND configuration. The engineer could navigate to Displays/AREA_A and select a display and select RUN. Conversely, when debugging a module instance the engineer can select operator display and the operator display will be run and allow the user to see the values from the display.

If the engineer decides to test an operator display instance the engineer can select "RUN" command and the display will be brought up in the "Operator Workspace", with values updating on the display. If the display has script or function blocks on it they can be debugged with breakpoints in the configuration workspace and the engineer can step through the script for an object or display event. When debugging an operator display instance the engineer can select the links to other operator displays as well as the linking to view online a control strategy that is associated with the object selected or the overall display selected.

Additionally, there is one other way the engineer can run a strategy or display online. The engineer can have a display class and select run. The user can then enter an instance and run that instance. Same thing should be possible for a module class.

3.9 Engineer exports the GS_BLEND package from DV System.

3.10 Engineer Imports the GS_BLEND package in a DV System.

Services Architecture

Table of Contents:

1	INTRODUCTION	275
1.1	STARBURN COMMUNICATION OBJECTIVES	275
1.2	SUPPORTED TOPOLOGIES.....	275
1.2.1	Small System	275
1.2.2	Large System.....	276
1.2.3	Multi-Zone	276
1.2.4	Remote Client	277
1.3	SCENARIOS	277
2	REQUIREMENTS.....	278
2.1	TOPOLOGY.....	278
2.1.1	Small	278
2.1.2	Large	278
2.1.3	Zones	278
2.1.4	Remote Client	278
2.2	SUPPORT CONFIGURATION CLIENTS TALKING TO SERVER	278
2.3	SUPPORT CONFIGURATION SERVICES	278
2.4	DIRECTORY SERVICE.....	278
2.5	CONNECTION MANAGER	279
2.6	REDUNDANCY.....	279
2.7	OBJECT BROKERING SERVICE.....	279
3	DATA SERVICES CONCEPT.....	280
3.1	OVERVIEW OF CONCEPT	280
3.2	SERVICE PROVIDERS	280
4	PROCESS GRAPHICS SCENARIOS	281
4.1	BACKGROUND	281
4.2	RUNTIME DATA SCENARIOS	282
4.2.1	Initialization Scenario	282
4.2.2	Load Display, Add Group & Item	283
4.2.3	Receive RT Data Change	286
5	CONFIGURATION SCENARIOS.....	288
5.1	BACKGROUND FOR SCENARIOS	288
5.1.1	DvDvServer Connection Model	288
5.1.2	Sample Hierarchy	289
5.1.3	Component Hierarchy	290
5.2	USER STARTS EXPLORER	291
5.2.1	Preconditions	291
5.2.2	Postconditions.....	291
5.2.3	Scenario	291
5.2.4	Component Sequence Diagram	291
5.3	NEXT SCENARIO.....	292
5.3.1	Preconditions	292
5.3.2	Postconditions.....	292
5.3.3	Scenario	292
5.3.4	Component Sequence Diagram	292
6	DATA SERVICES ARCHITECTURE.....	293
6.1	OVERVIEW	293
6.2	COMPONENT ARCHITECTURE	293

6.3	SUBSYSTEMS	294
7	COMMUNICATION SERVICES	295
7.1	OVERVIEW	295
7.2	COMMUNICATION FRAMEWORK	295
7.2.1	Description	295
7.3	COMMUNICATION SERVICES	296
7.3.1	Download Service	296
7.3.2	Upload Service	296
7.3.3	Passthrough Service	296
7.3.4	Commissioning Service	296
7.3.5	Debug Service	296
7.3.6	Diagnostic Service	296
7.3.7	Session Service	296
7.3.8	Licensing Service	296
7.3.9	Status Information Service	296
8	SESSION SERVICES	297
8.1	OVERVIEW	297
9	DIRECTORY	298
9.1	OVERVIEW	298
10	CONNECTION MANAGER	299
10.1	OVERVIEW	299
11	REDUNDANCY	300
11.1	OVERVIEW	300
12	OBJECT BROKERING SERVICE	301
12.1	OVERVIEW	301
13	COMMUNICATION SERVICES	302
13.1	OVERVIEW	302
14	COMMUNICATION SERVICES	303
14.1	OVERVIEW	303

1 Introduction

This document describes the communication architecture for the system.

1.1 Starburn Communication Objectives

1. Provide a Communication Infrastructure for all system applications
2. Support Connected as well as Wireless Connections
- 3.

1.2 Supported Topologies

1.2.1 Small System

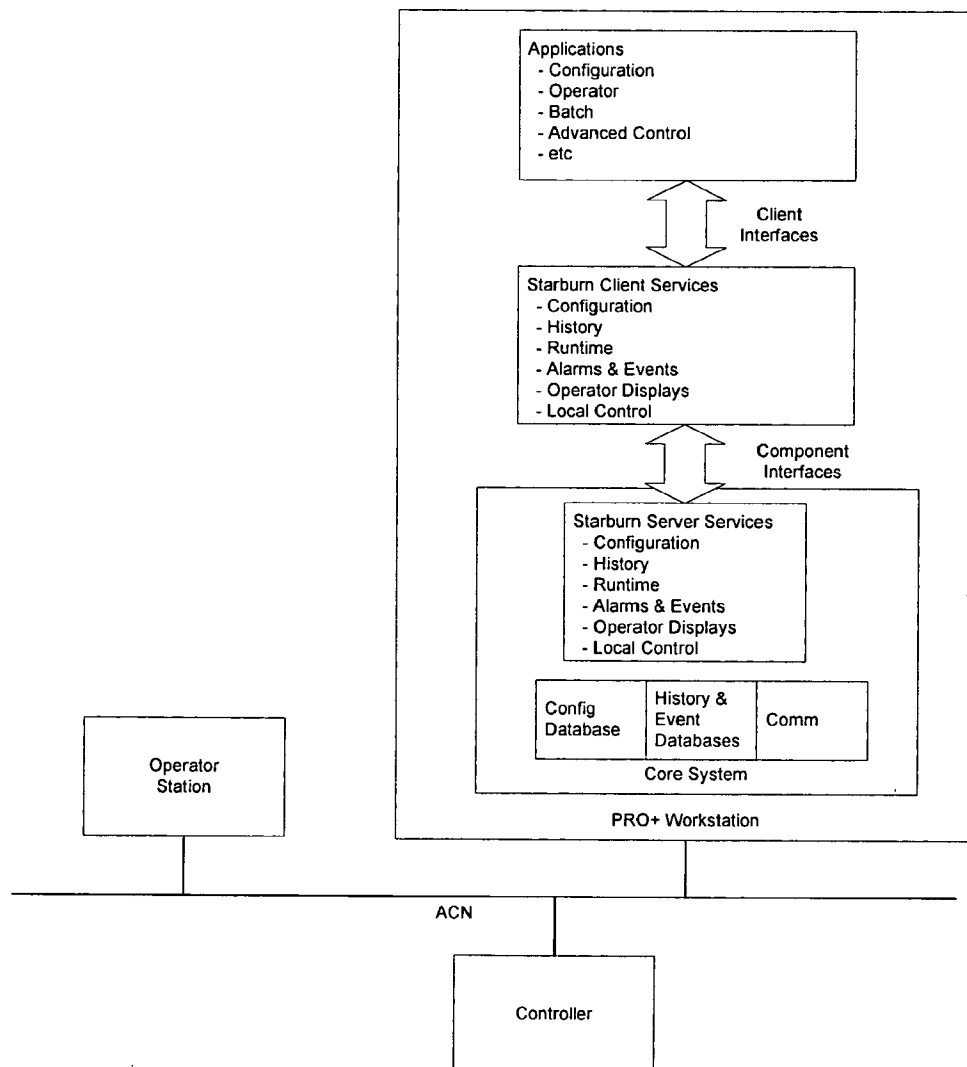


Figure 1. Small System

1.2.2 Large System

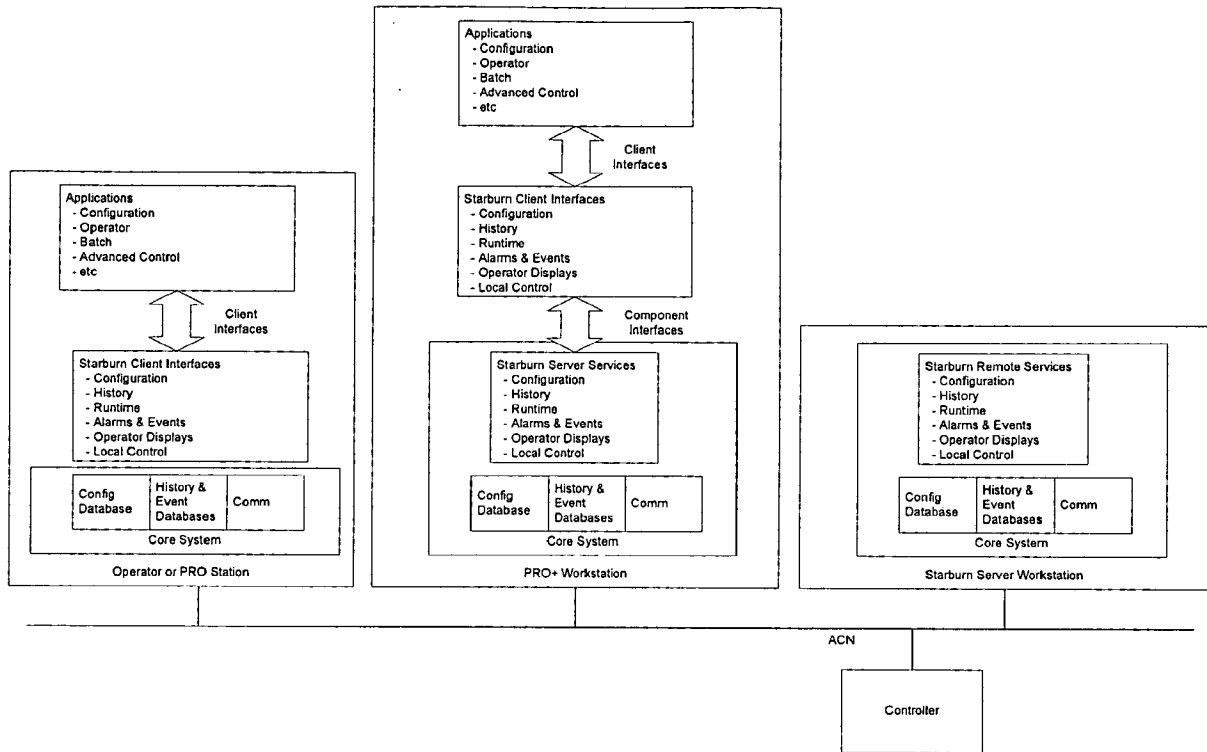


Figure 2. Large System

1.2.3 Multi-Zone

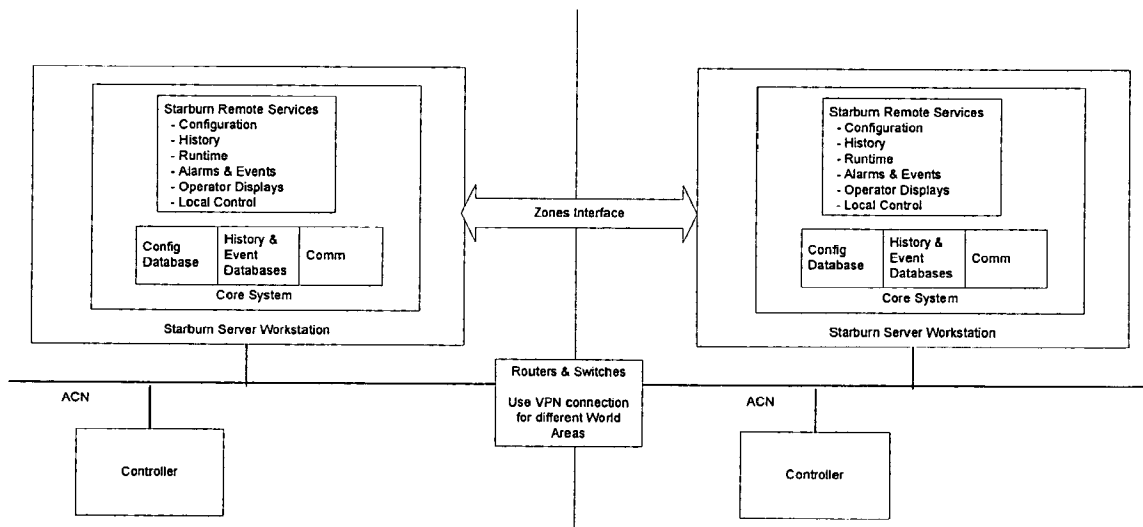


Figure 3. Multi-Zone System

1.2.4 Remote Client

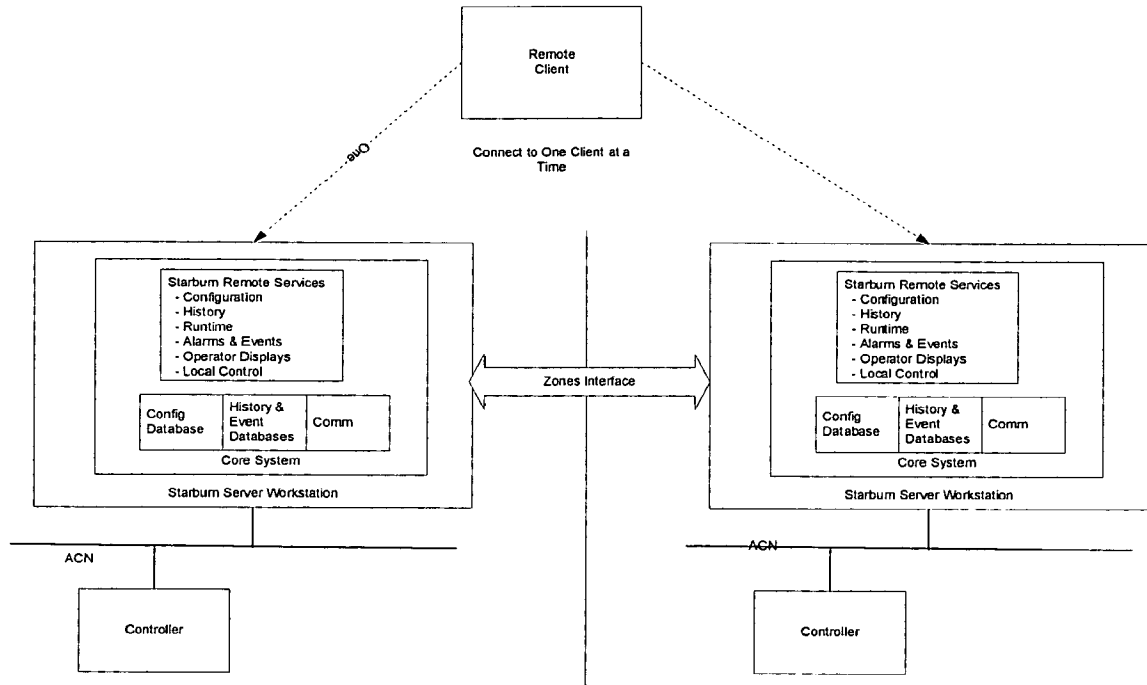


Figure 4. Remote Clients

1.3 Scenarios

1. Configuration
 - Connect to Server – Explorer to located and connects to DvDbServer
 - Expand Area 'AREA_A'
 - Updates are returned to a client
 - Starburn Server fails over from Active to Standby
2. Batch Executive
 - Save Graphic
3. Diagnostics

2 Requirements

2.1 Topology

2.1.1 Small

- Can host up to
 - 4 Starburn Client Nodes
 - 4 Controller or Old Style Operator Nodes

2.1.2 Large

- Can host up to
 - 120 total nodes (in any combination of the following)
 - 60 Starburn Client Nodes
 - 60 Old Style Operator Nodes
 - 100 Controller Nodes

2.1.3 Zones

- Can support up to 64 Zones

2.1.4 Remote Client

- Can connect to one Zone at a time
- May have a direct communication path to each Zone (hopping may not be supported)

2.2 Support Configuration Clients Talking to Server

- Local – on ACN (can use .NET Remoting)
- Remote – outside ACN (can connect to one System at a time)
- In another ACN (i.e. Zones)
 - Browsing (use .NET Remoting)
 - Transfer the configuration hierarchy (use .NET Remoting)
 - Full configuration (use Terminal Services)

2.3 Support Configuration Services

- Download (control strategies and displays)
- Upload (control strategies and displays)
- Status information (e.g. Node is downloaded)
- Commissioning
- Debug
- Diagnostic Information
- Session Support
 - Who is logged on
 - What rights does the logged on user have
- Licensing Support

2.4 Directory Service

- Object Location Transparency
 - Nodes
 - Modules
 - Displays
 - PI Data
 - Batch Data

-
- Localization Information

2.5 Connection Manager

- Forms connection over local networking
 - locally it would over our .NET channel service
 - between Zones it is over our .NET channel service
 - to Web devices it will be HPPT Post messages over SOAP (similar to what was done in the prototype)
- Report failed connections
- Report failover to backup services

2.6 Redundancy

- Communications switchovers (transparent to the client applications¹ – this is covered under “Object Location Transparency”, we change the source location of an object w/o the client having to be involved)
- Starburn Server Node switchovers (Active-to-Standby switchovers – may require clients to be notified and they in-turn to initiate some action such as “retry”)

2.7 Object Brokering Service

- Look up system services (e.g. DvDbServer is on node ‘X’, node ‘Y’ supports DvHistServer, etc)
- Given a service, what does it support.

¹ Communication switchovers may not be transparent to some of the applications. In some cases the application may need to know about the switchover.

3 Data Services Concept

3.1 Overview of Concept

Starburn will provide low level services that can be used by multiple data providers. The low level services take care of connections, sessions, redundancy, retries, timeouts, etc. The general model for these services is summarized below:

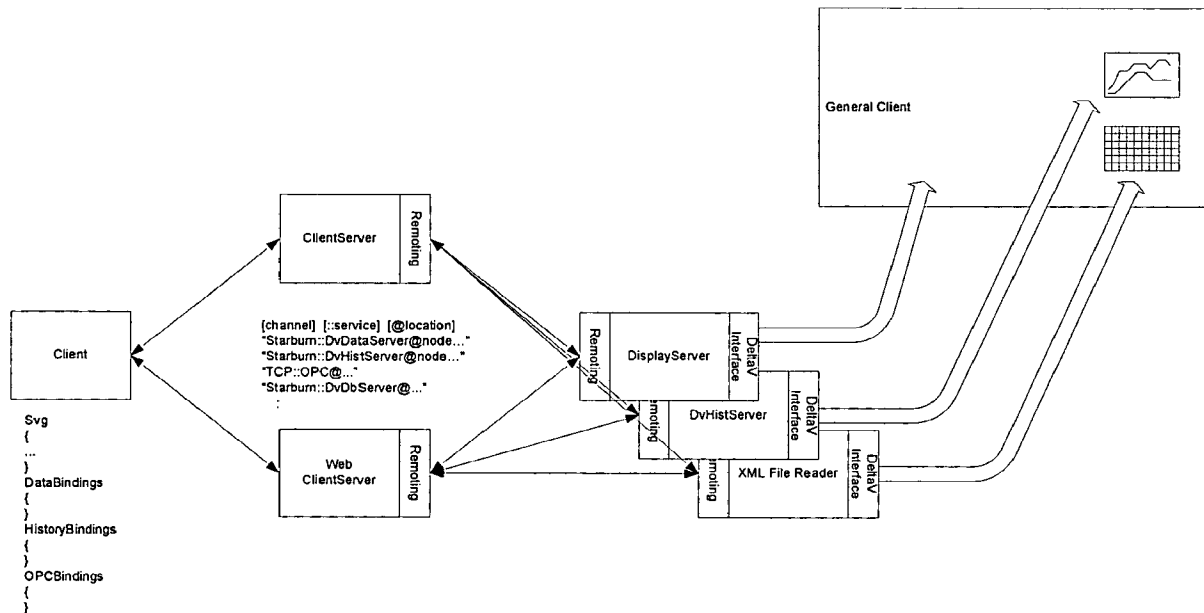


Figure 5. Integrating Multiple Data Providers

3.2 Service Providers

The service providers for the Clients will consist of:

- 1- Runtime Data
- 2- Historian
- 3- Alarms
- 4- Events
- 5- OPC Data Services
- 6- XML Files
- 7- HTTP Links

Each of these services will have a very specific interface defined for it. System Services implementing these services will follow the service interfaces. This allows clients to remain constant even when services are swapped out. For example, as part of the DeltaV and Ovation installations, the Service Providers will be installed.

4 Process Graphics Scenarios

4.1 Background

The Process Graphics Display environment supports interfacing with multiple data services. This was discussed in the data services concept. For Process Graphic Displays the support would be provided as illustrated below:

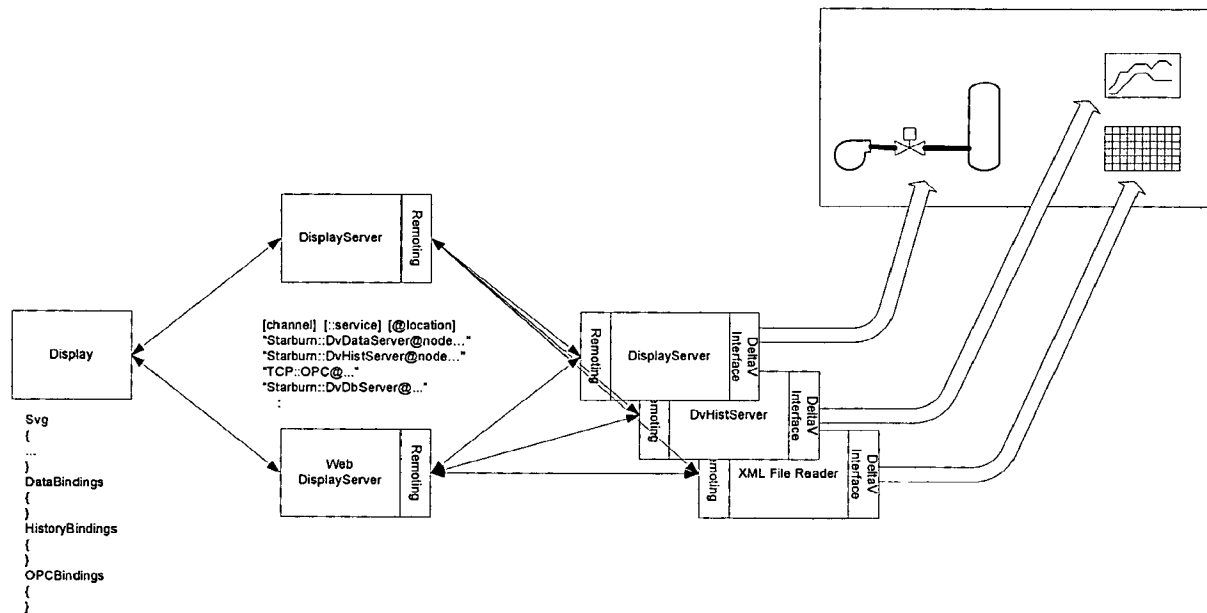


Figure 6. Integrating Process Graphics With Multiple Data Providers

4.2 Runtime Data Scenarios

4.2.1 Initialization Scenario

4.2.1.1 Preconditions

- 1- DeltaV Services Started
- 2- Display Package Loaded
- 3- User not logged in

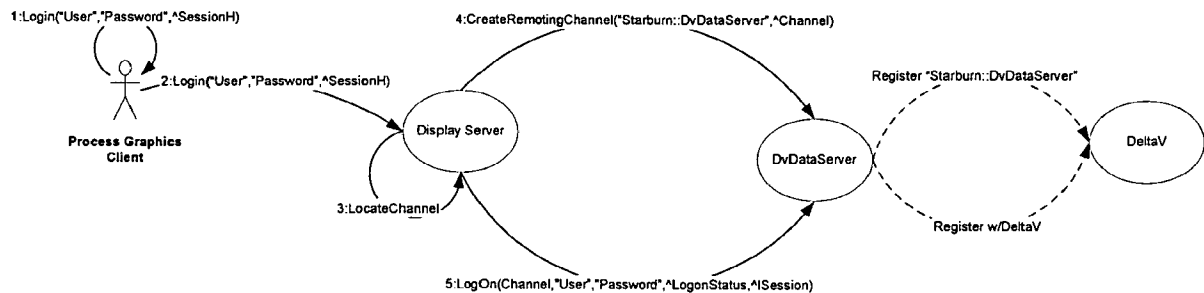
4.2.1.2 Postconditions

- 1- User Logged in

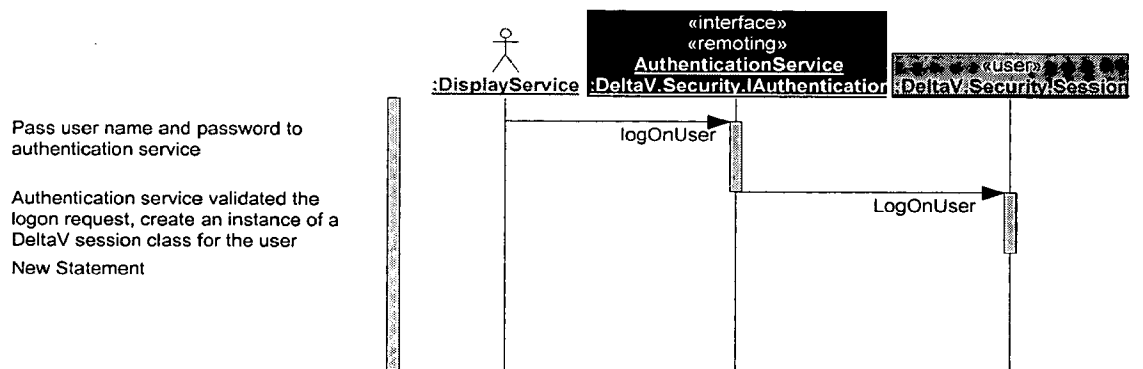
4.2.1.3 Scenario

- 1- User selects option to log into DeltaV
 - a. A dialog is presented to log into DeltaV
 - b. System prompts for User Name and Password
 - c. User enters User Name and Password
- 2- Process Graphics Client requests a session handle by calling "Login("User","Password",^SessionH) on Display Server
- 3- Display Server checks for an existing channel by calling LocateChannel
- 4- Display Server creates a channel to DvDataServer by calling CreateRemotingChannel("Starburn::DvDataServer",^Channel) on DvDataServer
- 5- Display Server requests a session for the user by calling LogOn(Channel,"User","Password",^ISession)

4.2.1.4 Object Collaboration Diagram



4.2.1.5 Component Sequence Diagram



4.2.2 Load Display, Add Group & Item

4.2.2.1 Preconditions

- 1- Process Graphics Package Loaded
- 2- User has a session handle allocated
- 3- User has just browsed and requested Process Graphics Client to load Display "DISP1"

4.2.2.2 Postconditions

- 1- Display "DISP1" loaded
- 2- Data Fields in "DISP1" have been bound

4.2.2.3 Scenario

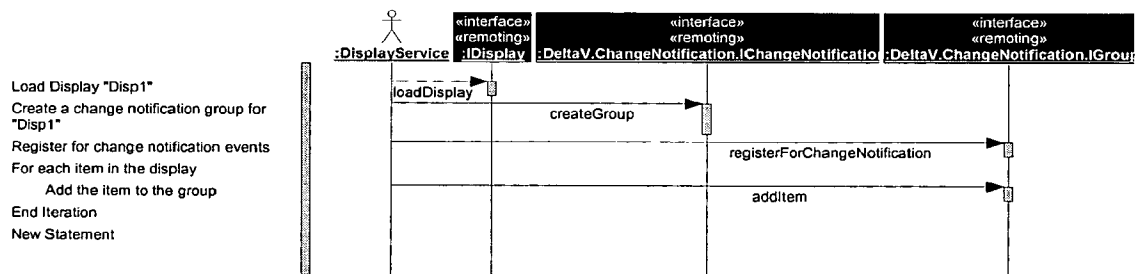
- 1- Process Graphics Client requests display to be loaded by calling LoadDisplay(SessionH,"DISP1",^SVG) on Display Server
- 2- Display Server requests a Display Service by calling CreateDisplayInterface(ISession,^IDisplayService) on Display Server
- 3- Display Server requests display by calling ReadDisplay(ISession,"DISP1",^XML)
- 4- Process Graphics Client processes XML file. Rt Data Bindings are bound by calling AddGroupsAndItems(SessionH,RtDataBindings,^InitialRtValues) on Display Server

- 5- Display Server sends request AddGroup(UpdateRate,SingleFields,^IGroup) to DvDataServer
- 6- DvDataServer sets up DeltaV communications
 - a. Session::Locate(^RTSecureModule)
 - b. CreateAccessToken(^AccessToken)
 - c. OCN::BindAccessToken(^RT_SUCCESS)
- 7- DvDataServer adds items to group by calling AddItems({items},^{ItemH})
- 8- History Data Bindings are bound by calling AddHistorytems(SessionH,HistDataBindings,^InitialHistValues) on Display Server
- 9- Display Server sends request AddHistoryItems({HistDataBindings},^{Items}) to DvHistServer
- 10- XML File Bindings are bound by calling AddXMLItems(SessionH,XMLFileDataBindings,^InitialFileValues) on Display Server
- 11- Display Server sends request AddHistoryItems({HistDataBindings},^{Items}) to DvHistServer
- 12- Process Graphics Client Renders Display

4.2.2.4 Object Collaboration Diagram



4.2.2.5 Component Sequence Diagram



4.2.3 Receive RT Data Change

4.2.3.1 Preconditions

- 1- Group and Data Items has been set up
- 2- An item has changed in DeltaV

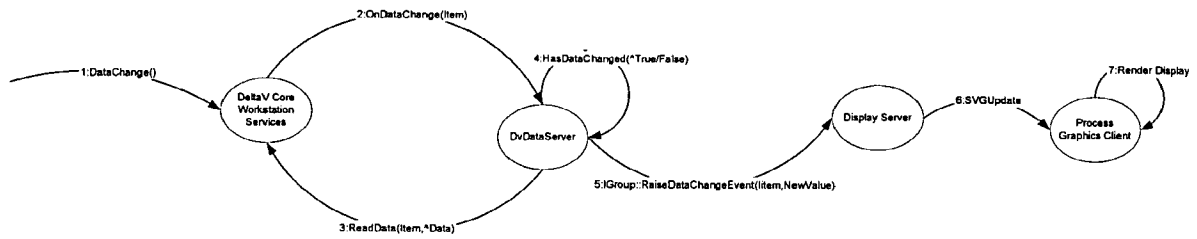
4.2.3.2 Postconditions

- 1- Update has been processed

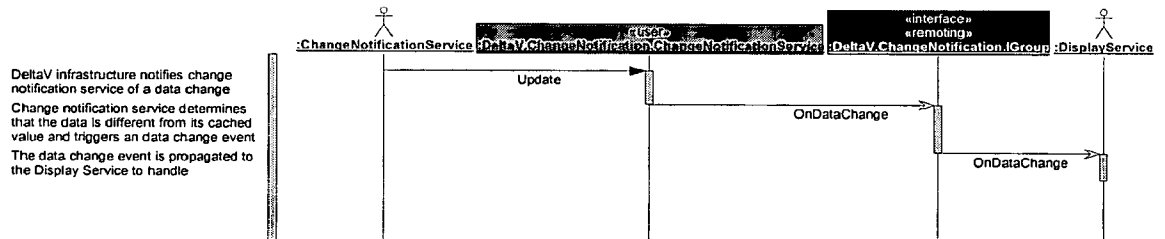
4.2.3.3 Scenario

- 1- DeltaV System sends a DataChanged() request to DeltaV Core Workstation Services
- 2- DeltaV Core Workstation Services sends a OnDataChange(Item) notice to DvDataServer
- 3- DvDataServer gets current data value by calling ReadData(Item, ^Data) on DeltaV Core Workstation Services
- 4- DvDataServer checks to see if data value has really changed
- 5- If the data has changed a event is raised by calling
IGroup::RaiseDataChangeEvent(litem, NewValue) on DisplayServer
- 6- DisplayServer passes data value on to Process Graphics Client
- 7- Process Graphics Client Renders Display

4.2.3.4 Object Collaboration Diagram



4.2.3.5 Component Sequence Diagram



5 Configuration Scenarios

5.1 Background for Scenarios

5.1.1 DvDvServer Connection Model

The connection model used by Starburn will contain the services illustrated in the following drawing. The exact methods and naming are being worked out.

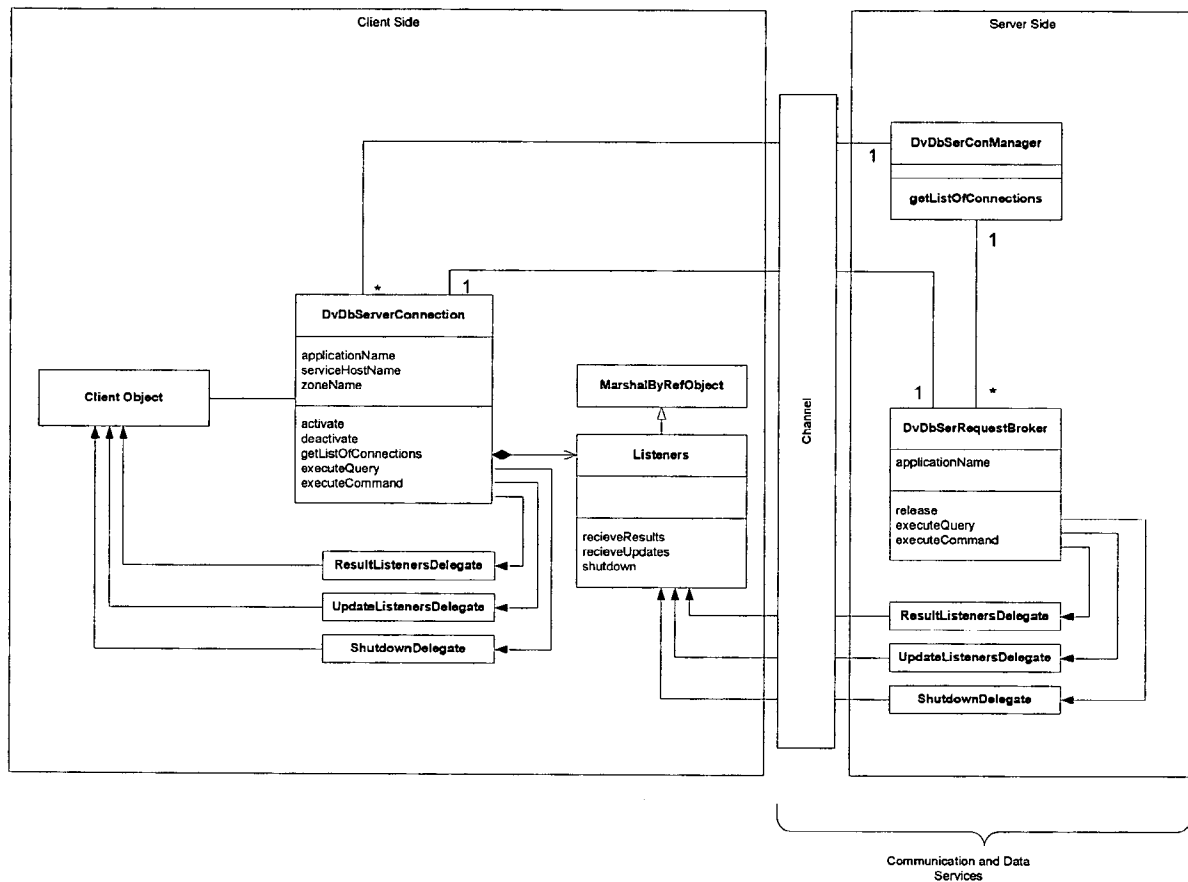


Figure 7. DvDbServer Service Connections

5.1.2 Sample Hierarchy

Several of the scenarios make use of the following hierarchy:

Site

- + Library
- + System Configuration
 - + Recipes
 - + Setup
 - + Control Strategies
 - + AREA_A (Area)
 - + LINE_1 (Process Cell)
 - + BOILER_A (Unit)
 - + WATER_INLET_FLOW (Equipment Module)
 - + VALVE_1 (Module)
 - + FLOW_METER_1 (Module)
 - AI1 -> FFAI1
 - + PLM_1 (Phase Logic Module)
- + Physical Network
 - + Control Network
 - + CTRL1
 - + I/O
 - + C01
 - + CH01
 - + CH02
 - :
 - + C01
 - + P01
 - + PDT1
 - FFAI1
 - FFAI2

5.1.3 Component Hierarchy

The scenarios in the following sections make use of several new components and interfaces. These are illustrated below.

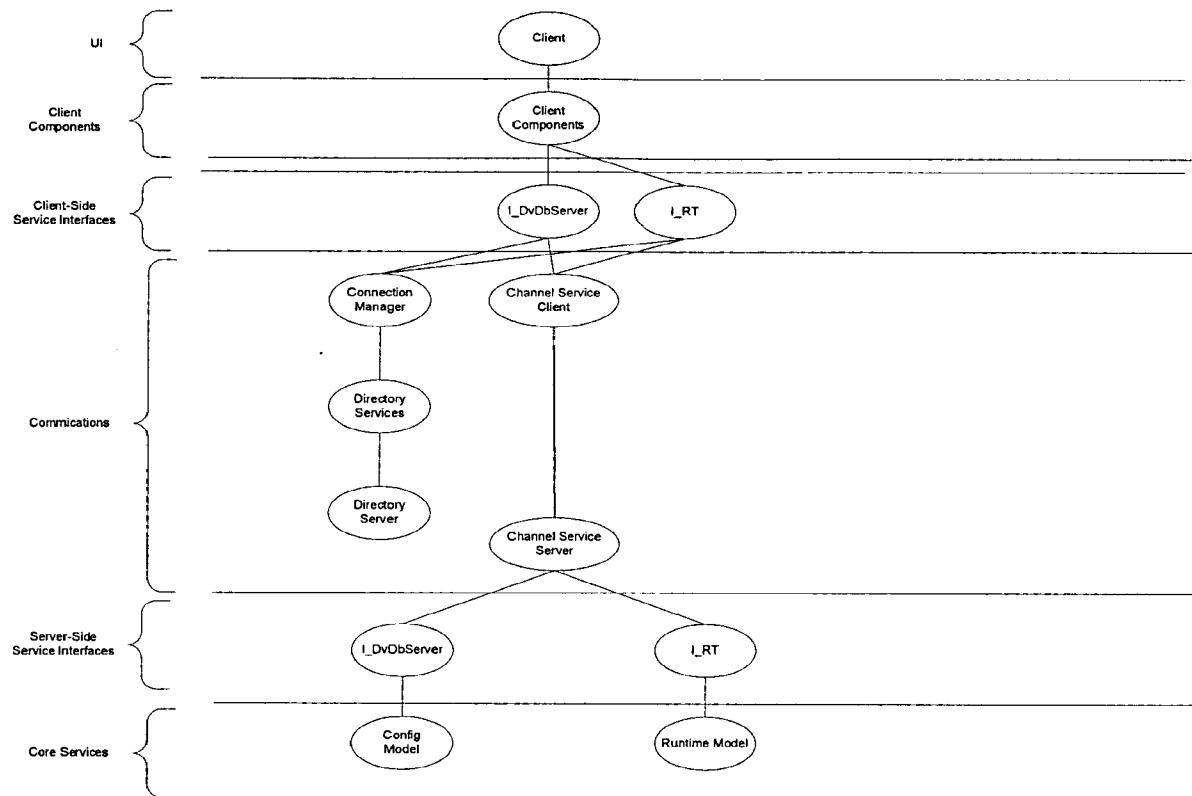


Figure 8. Component Layers

5.2 User Starts Explorer

5.2.1 Preconditions

- 1- Explorer is not started in 'this' node
- 2- No other configuration applications are running on this node
- 3- A Starburn Server exists

5.2.2 Postconditions

- 1- Explorer has been started
- 2- A .NET Remoting channel has been set up between the Client 'EXP' and the Server 'DvDbServer'

5.2.3 Scenario

- 1- User selects Start Menu->Starburn->Engineering->Explorer (*not sure what the navigation will ultimately be, for now assuming classical navigation*)
- 2- Client 'Exp' creates Connection Object 'DvDbServer_Client' (probably executing something like `DvDbServer_Client = new DvDbServerServerConnection(string "Zone", string "SB_PRO+", service "DvDbServer")`)
 - a. Connection Object 'DvDbServer_Client' requests a channel from Connection Manager 'ClientConnectionManager'
 - b. Connection Manager 'ClientConnectionManager' contacts Directory Services to locate service 'DvDbServer'
 - c. Directory Services returns address information for service 'DvDbServer'
 - d. Connection Manager 'ClientCM' sends message to Connection Manager 'PROPlusCM' to allocate channel
 - i. Connection Manager 'PROPlusCM' allocates channel 'CH_1'
 - ii. Connection Manager 'PROPlusCM' returns channel 'CH_1' to Connection Manager on 'ClientCM'
 - e. Connection Manager returns channel 'CH_1' to Connection Object 'DvDbServer_Client'
- 3- Client 'Exp' activates Server Connection (probably by executing something like `DvDbServer_Client.Activate(channel "CH_1")`)

5.2.4 Component Sequence Diagram

5.3 Next Scenario

5.3.1 Preconditions

1- E

5.3.2 Postconditions

1- A

5.3.3 Scenario

1- U

5.3.4 Component Sequence Diagram

6 Data Services Architecture

6.1 Overview

The technical architecture for data services delivers the technical components and frameworks that support what we are trying to build.

6.2 Component Architecture

The drawing below illustrates the initial component map after the exercises executed above. Packages have been added to represent the logical components. Most components have an interface added to provide services.

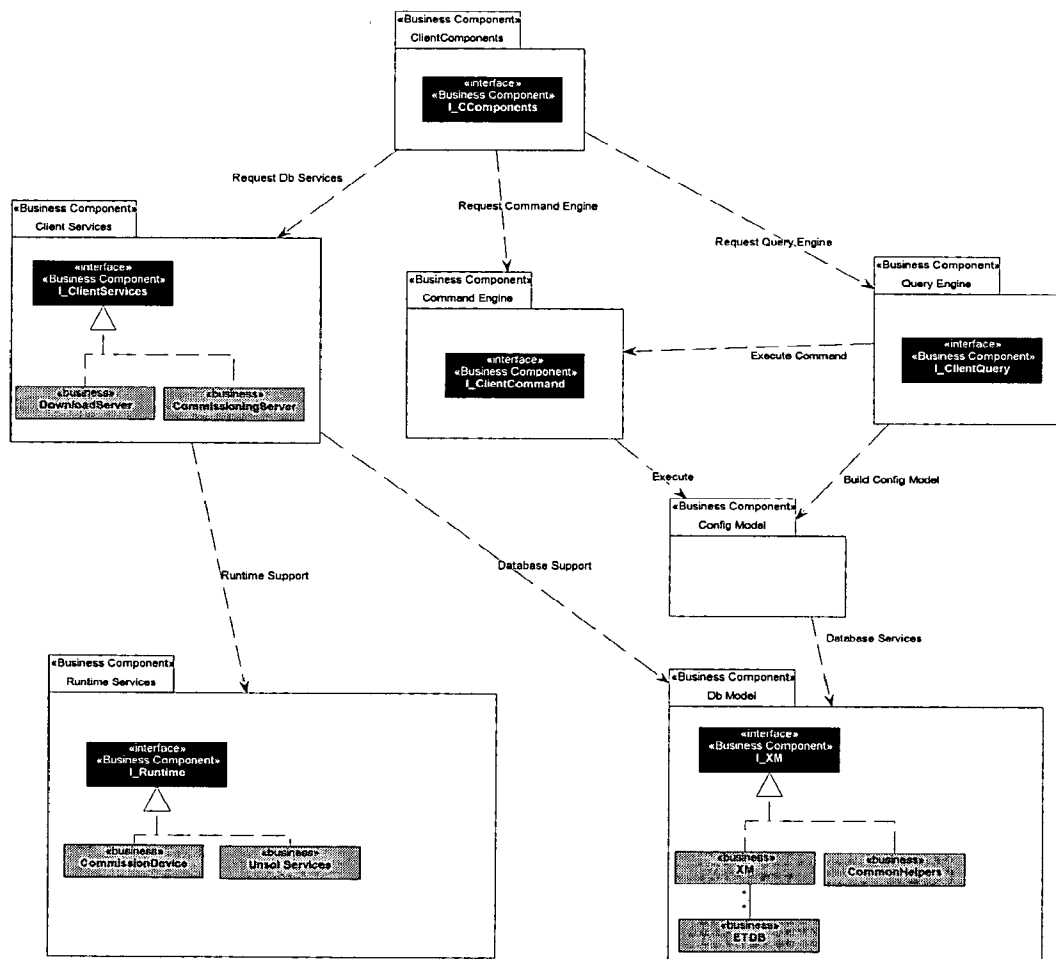


Figure 9. Component Architecture

6.3 Subsystems

Subsystems provide a high-level view of how the architecture realizes Use Cases without going into the low-level details. The concept of subsystems sometimes carries through to implementation in which case real components implement the subsystem.

The subsystems below are a partial list. The list summarizes what I thought about as I assembled the minutes from the architecture document.

Subsystem Name	Subsystem Description
Operator Workspace	Provides a controlled environment for Process Graphics, Batch Operator Interfaces, Advanced Control Applications, Alarm Banner, Alarm Summary, Faceplates, Detail displays, History Clients, etc.
Engineering Workspace	Provides a controlled environment for Configuring Process Graphics, Batch, Advanced Control, Alarms, Displays, Devices, History, etc.
Process Graphics	Provide user defined views and operator interfacing onto the system architecture.
UI	Provides Specific Controls, Frameworks for application developers, UI design guidelines, etc
Configuration	Provides configuration services for both connected and disconnected clients. Connected clients include applications to configure control strategies, displays, IO, user accounts, manage configuration, load licenses, etc. Disconnected clients are used for bulk editing the system, for interfacing to external tools such as INtools, and for addressing customer specific requirements with respect to Recipe Management, etc. Also includes services for Versioning and Audit Trail.
Communication	Provides communication services for control, debugging, configuration, alarm & alerts, operator interfaces, etc.
Data Services	Provides low level access to the system. This consists of things such as the Custom Function Block interface for interfacing foreign DCS systems (e.g. Bailey, Teleperm),
Session	Provide support for user logon, session management, secure access to the system, and span-of-control.
Events / Alarm / Alert Services	Allows monitoring and notification of significant system states, acknowledgements, and priority calculations.
Licensing	Provide services to enable functionality on the system using licensing.
Diagnostics	Interacts with all of the subsystems to collect diagnostic and alert information.
System Administration Services	Provides services for installing applications and hot fixes. Also provides support for remotely monitoring and supporting systems.
History	Provides access to continuous, event, and batch data.
Control	Provides continuous, sequencing, batch, recipe, process, safe control.
PIO	Provides access to all of the IO – including conventional IO, Fieldbus, Profibus, DevicNet, Serial, ASi, Remote IO, RS3 IO, Provox IO, Safe IO, etc.
Report Writer	Provides reporting services for alarm/alert information, document configuration, etc.

7 Communication Services

7.1 Overview

The communications subsystem with it's supporting frameworks are illustrated below.

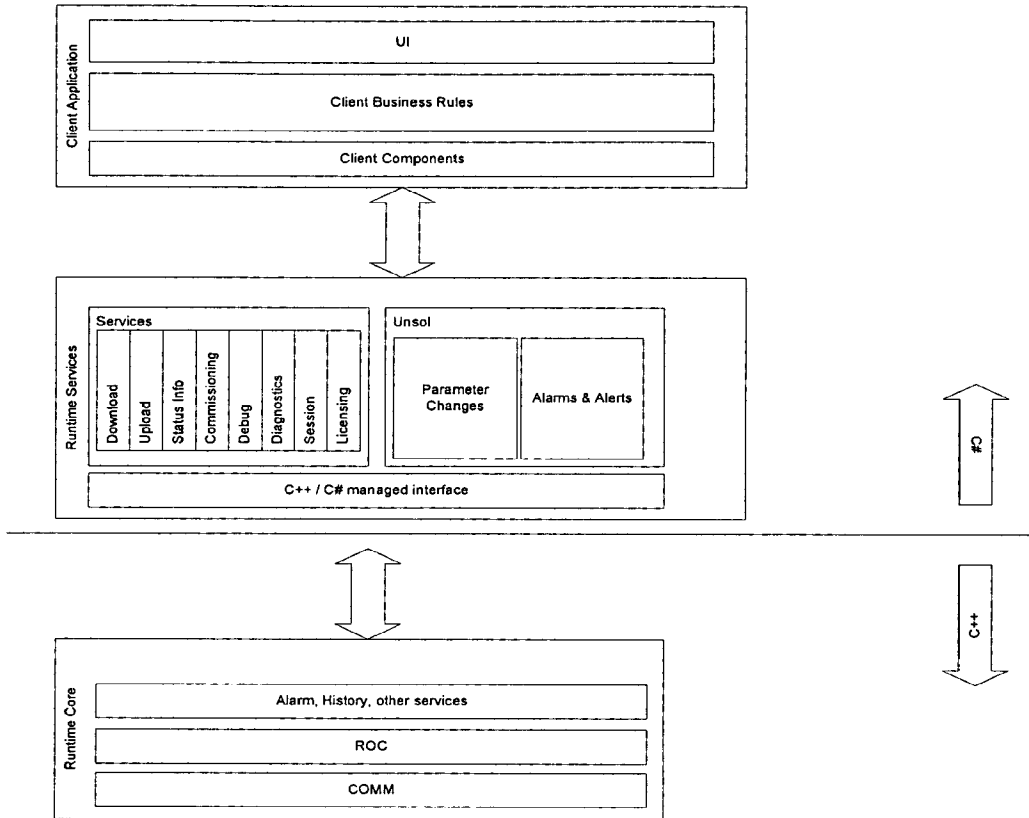


Figure 10. Deployment View of Communication Framework

The ...

7.2 Communication Framework

7.2.1 Description

The biggest change for clients is the introduction of the communication assemblies and components...

The communication framework is shown below...

Figure 11. Communication Model

The Configuration Model in more detail is shown below.

Figure 12. Configuration Model

7.3 Communication Services

7.3.1 Download Service

7.3.2 Upload Service

7.3.3 Passthrough Service

7.3.4 Commissioning Service

7.3.5 Debug Service

7.3.6 Diagnostic Service

7.3.7 Session Service

7.3.8 Licensing Service

7.3.9 Status Information Service

8 Session Services

8.1 Overview

The

29

9 Directory

9.1 Overview

The

10 Connection Manager

10.1 Overview

The

11 Redundancy

11.1 Overview

The

12 Object Brokering Service

12.1 Overview

The

13 Communication Services

13.1 Overview

The communications subsystem with it's supporting frameworks are illustrated below.

14 Communication Services

14.1 Overview

The communications subsystem with it's supporting frameworks are illustrated below.

System Globalization and Localization

Table of Contents:

1	INTRODUCTION	306
1.1	OVERVIEW	306
1.2	DEFINITIONS.....	307
1.2.1	Globalization	307
1.2.2	Culture/Locale.....	307
1.2.3	Neutral Culture.....	307
1.2.4	Specific Culture.....	307
1.2.5	Localizability.....	307
1.2.6	Localization	307
1.2.7	Resources.....	308
1.2.8	Satellite Assemblies.....	308
1.3	REFERENCES	308
2	OBJECTIVES.....	308
3	GLOBALIZING THE APPLICATION.....	308
3.1	LOCALIZING STRING MESSAGES.....	308
3.2	LOCALIZING EXCEPTION MESSAGES	310
3.3	DVRESOURCEGENERATOR.....	313
3.4	SUMMARY	316
4	LOCALIZATION SUPPORT	316
4.1	EXTRACTING THE DEFAULT RESOURCES	316
4.2	TRANSLATING THE RESOURCES	317
4.3	CREATING THE LOCALIZED SATELLITE ASSEMBLIES.....	318
5	APPENDIX – GLOBALIZATION CLASSES	319
5.1	RESOURCESTRINGS CLASS	319
5.2	STRINGRESOURCEATTRIBUTE CLASS.....	320
5.3	CONFIGEXCEPTION CLASS.....	320

1 Introduction

1.1 Overview

The .NET Framework provides extensive support for the development of world-ready applications. The .NET localization model consists of a main assembly that contains both the application code and the default resources (strings, images, objects, etc) in English. Each localized application will have satellite assemblies for each specific locale that will be supported by the application. In the localization model, resource fallback, or the loading of the corresponding resources, happens in a hierarchical manner.

- At the top of the hierarchy sit the default resources contained in the main assembly. This is the eventual catch all for resource searches.
- Below the default resources are the resources for any neutral cultures.
- Below the neutral culture resources are the resources for any specific culture.

When a localized resource such as a string is loaded and is not found, the hierarchy will be traversed until a resource file containing the requested resource is found. This is best illustrated by the figures below:

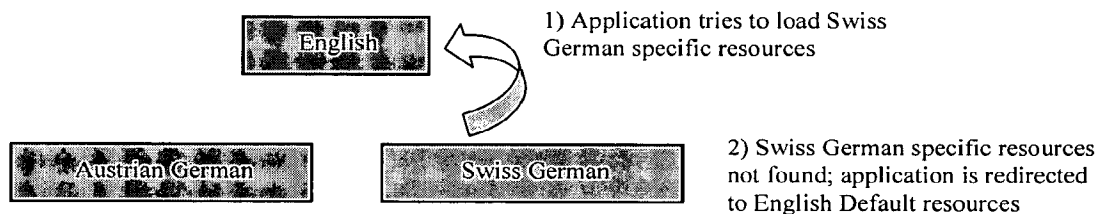


Figure 1.1.1. Neutral culture not specified

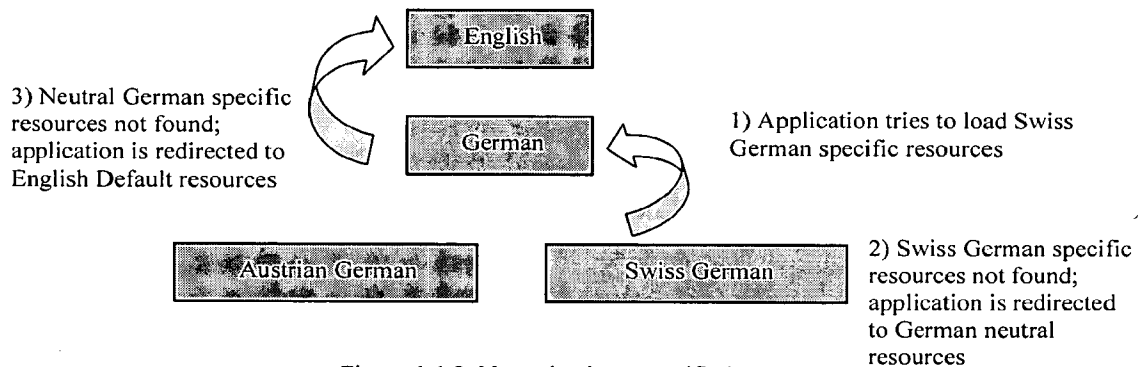


Figure 1.1.2. Neutral culture specified

Since the localized satellite assemblies contain resources and no code at all, they can be compiled and translated at any time. Translation could be done parallel to development or even after the application has shipped out.

The Starburn project will leverage on this localization model by providing a localization framework that will be followed throughout the project's lifetime. The Starburn modules will be designed and developed as a globalized application and localizability testing will be integrated into the testing framework. Finally, a localization toolkit will be provided to allow clients and/or third-party translators to do the actual localization or translation work.

NOTE: The rest of this document will be focusing on the localization framework provided for back-end applications. UI localization will be deferred to a later time when the configuration workspace concept has developed to a more mature stage. The localization toolkit will also be assembled when all tools and procedures are made available.

1.2 Definitions

1.2.1 Globalization

Globalization is the process of designing and developing a software product that functions in multiple cultures/locales. This process involves:

- Identifying the culture/locale that may be supported
- Designing features which support those cultures/locales
- Writing code that functions equally well in any of the supported cultures/locales

In other words, globalization adds support for input, display, and output of a defined set of language scripts that relate to specific geographic areas. The most efficient way to globalize these functions is to use the concept of cultures/locales.

1.2.2 Culture/Locale

A culture/locale is a set of rules and a set of data that are specific to a given language and geographic area. These rules and data include information on:

- Character classification
- Date and time formatting
- Numeric, currency, weight, and measure conventions
- Sorting rules

1.2.3 Neutral Culture

A neutral culture is associated with a language but not a region. For example, English ("en") is a neutral culture.

1.2.4 Specific Culture

A specific culture is associated with a language and a region. For example, English United States ("en-US") is a specific culture.

1.2.5 Localizability

Localizability is the ability of a globalized application and/or content (including text and non-text elements) to be adapted for any local market (locale).

1.2.6 Localization

Localization is the process of adapting a globalized application and/or content, which you have already processed for localizability, to meet the language, cultural and political expectations and/or requirements of a specific local market (locale).

The localization process refers to translating the application user interface (UI) or adapting graphics for a specific culture/locale. The localization process can also include translating any help content associated with the application.

1.2.7 Resources

Resources are anything that the application uses to operate that is not considered executable code. Examples of resources include icons, graphic files, strings and objects.

1.2.8 Satellite Assemblies

Satellite assemblies are .NET assemblies that contain resources (strings, graphics, objects, etc) . These are typically associated with a main assembly that contains the code that uses the resources available in the satellite assemblies.

1.3 References

The localization model for the .NET Framework is discussed more thoroughly in MSDN.

2 Objectives

There are two basic objectives for this prototype:

- Provide a framework for developers to enable localization and globalize the application with minimal impact on coding effort and time.
- Provide a tool for localizing default resources contained in an assembly.

3 Globalizing the Application

The idea is to enable localization support with minimum effort from the developers. Since the localization routines are pretty much standard, a framework could be developed to help developers with globalizing their application. This framework leverages on existing work done for the RoadRunner project. The existing localization classes were re-structured to provide a more streamlined code. The framework currently handles two major items, string messages and exception messages.

3.1 Localizing String Messages

The .NET Framework contains a `ResourceManager` class that provides convenient access to culture-specific resources at runtime. The `ResourceManager` class is encapsulated in a `ResourceStrings` class to provide specific methods in accessing localized resources in Starburn. The `ResourceStrings` class is realized in the `DeltaV.Globalization` namespace and provides the following:

An overloaded `GetString` method to enable a more efficient and simplified access to localized strings.

A list of resource managers for each resource file. This list is maintained and automatically loaded when a resource is requested. This frees the developer from instantiating resource managers in their codes.

The `ResourceStrings` class in turn is inherited by corresponding resource classes. This means that each resource file will have a corresponding .NET class that will be added to the application. For example, a resource file `ClientModelSchema.resx` will have a corresponding class called `ClientModelSchema.cs`. This provides a means of treating each resource identifier as an object with a public interface for retrieval of the localized string itself. Some additional benefits of the resource class are as follows:

- Incorrect number of arguments when formatting resource strings is caught at compile time.
- IntelliSense supplies the retrieval method name and reminds you of any parameters for formatted strings.

For example, given a resource file called `ClientModelSchema.resx`

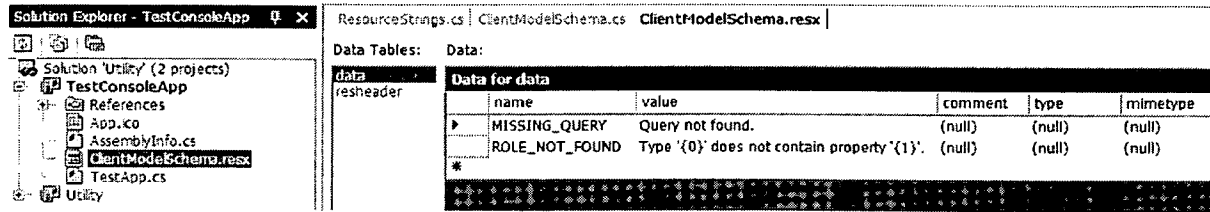


Figure 3.1.1. Sample string resource file

A corresponding class (`ClientModelSchema.cs`) would contain the following:

```
using System;
using System.Reflection;
using DeltaV.DataAccess.Utility;

namespace TestConsoleApp
{
    /// <summary>
    /// Provides access to an assembly's string resources.
    /// </summary>
    [StringResource( "TestConsoleApp.ClientModelSchema", typeof(ClientModelSchema) )]
    public class ClientModelSchema : ResourceStrings
    {
        static private string _assemblyName;

        static ClientModelSchema()
        {
            _assemblyName = typeof(ClientModelSchema).AssemblyQualifiedName ;
        }

        #region Localized Resources

        internal static string MISSING_QUERY()
        {
            return ResourceStrings.GetString( _assemblyName, "MISSING_QUERY" );
        }

        internal static string ROLE_NOT_FOUND( object arg0, object arg1 )
        {
            return ResourceStrings.GetString( _assemblyName, "ROLE_NOT_FOUND", arg0, arg1 );
        }

        #endregion
    }
}
```

Figure 3.1.2. Sample resource class

The class uses a custom attribute called `StringResource` that will provide resource file information to the `ResourceStrings` class. This enables the `ResourceStrings` class to determine which `ResourceManager` to use in accessing resource files. Each resource in the resource file is accessed thru a method. The string formatting contained in the resource file will be specified as method parameters. In the given example,

`ROLE_NOT_FOUND = "Type '{0}' does not contain property '{1}'."`

The string formatting specified (`{0}` and `{1}`) are converted into parameters (`object arg0`, `object arg1`) in the method. This effectively enables the Intellisense support in Visual Studio. Figure 3.1.3 shows a snapshot of the Visual Studio editor that demonstrates the listing of resources available from a given class.

```

class TestApp
{
    /// <summary>
    /// The main entry point for the application.
    /// </summary>
    [STAThread]
    static void Main(string[] args)
    {
        // Set the locale to German Switzerland
        Thread.CurrentThread.CurrentCulture = CultureInfo.CreateSpecificCulture("de-CH");

        Console.WriteLine( "Resource1 = " + ClientModelSchema.MISSING_QUERY() );
        Console.WriteLine( "Resource2 = " + ClientModelSchema.ROLE_NOT_FOUND(
    }
}

```

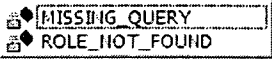


Figure 3.1.3. IntelliSense provides a list of resources available

Figure 3.1.4 on the other hand shows how the parameter list shows up in IntelliSense. This helps the developer in providing parameters to localized resource strings that contains string formatting characters.

```

Console.WriteLine( "Resource1 = " + ClientModelSchema.MISSING_QUERY() );
Console.WriteLine( "Resource2 = " + ClientModelSchema.ROLE_NOT_FOUND(
    string ClientModelSchema.ROLE_NOT_FOUND( object arg0, object arg1)
}

```




Figure 3.1.4. IntelliSense provides a parameter list

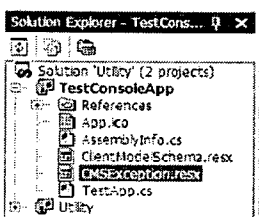
Figures 3.1.3 and 3.1.4 also shows how easy it is to access localized strings from within the code.

A potential drawback to this approach is the fact that for each resource string, a corresponding method needs to be written. This could potentially lead to a lot of methods and hand coding the resource classes would be very tedious indeed. Not to mention that the string formatting will have to be checked accurately to ensure that the method reflects the correct number of parameters. This issue will be discussed in more details in section 3.3.

3.2 Localizing Exception Messages

A natural extension to the localization of string messages is the localization of exception messages. The general idea with localizing exception messages is to encapsulate the localization of string messages and then add methods by which the actual exceptions can be raised.

Consider the exception resource file below:



Data for data				
name	value	comment	type	mimetype
LCS_INVALID_XML	Error parsing XML. The value '{0}' is not valid.	(null)	(null)	(null)
LCS_MISSING_END_TAG	Missing end tag in XML.	(null)	(null)	(null)

Figure 3.2.1. Sample Exception Resource File

Notice that this is just another .NET resource file. We distinguish an exception resource thru the name . Exception resources always end with the word "Exception" like in the example above. A corresponding class to this exception resource file would look like the following:

```

using System;
using System.Reflection;
using System.Runtime.Serialization;
using DeltaV.DataAccess.Utility;

namespace TestConsoleApp
{
    /// <summary>
    /// Provides access to an assembly's string resources.
    /// </summary>
    [StringResource( "TestConsoleApp.CMSException", typeof(CMSExceptionResource) )]
    internal class CMSExceptionResource : ResourceStrings
    {
        static private string _assemblyName;

        static CMSExceptionResource()
        {
            _assemblyName = typeof(CMSExceptionResource).AssemblyQualifiedName ;
        }

        #region Localized Resources

        internal static string LCS_INVALID_XML( object arg0 )
        {
            return ResourceStrings.GetString( _assemblyName, "LCS_INVALID_XML", arg0 );
        }

        internal static string LCS_MISSING_END_TAG()
        {
            return ResourceStrings.GetString( _assemblyName, "LCS_MISSING_END_TAG" );
        }

        #endregion
    }

    /// <summary>
    /// A special resource class for handling the raising and displaying of exception messages.
    /// </summary>
    [Serializable]
    public class CMSException : ConfigException
    {
        public CMSException( string exceptionMessage ) : base ( exceptionMessage ) {}
        public CMSException( string exceptionMessage, Exception exception ) : base ( exceptionMessage, exception ) {}
        public CMSException( SerializationInfo info, StreamingContext context ) : base ( info, context ) {}

        internal static void Raise_LCS_INVALID_XML(Exception ex, object arg0 )
        {
            Raise( typeof(CMSException), CMSExceptionResource.LCS_INVALID_XML(arg0), ex );
        }
        internal static void Raise_LCS_INVALID_XML( object arg0 )
        {
            Raise( typeof(CMSException), CMSExceptionResource.LCS_INVALID_XML(arg0) );
        }

        internal static void Raise_LCS_MISSING_END_TAG(Exception ex)
        {
            Raise( typeof(CMSException), CMSExceptionResource.LCS_MISSING_END_TAG(), ex );
        }
        internal static void Raise_LCS_MISSING_END_TAG()
        {
            Raise( typeof(CMSException), CMSExceptionResource.LCS_MISSING_END_TAG() );
        }
    }
}

```

Figure 3.2.2. Sample Exception Class

The first thing you will notice is that there are now two classes. The first class is actually the same as the class created when localizing string messages. The noticeable difference is that this time, this class is now internal as opposed to being public. Other than that, it is just a localized string message class. The second class is what makes this more interesting. First off, this class inherits a class called `ConfigException`. This class is also available from the `DeltaV.Globalization` namespace and provides the following:

- An overloaded `Raise` method for raising an application exception.
- An overloaded `ToString` method to enable chaining of exception messages. That is, messages from inner exception are also retrieved and all the messages are then enumerated as a list.

Next, each exception resource gets associated with a corresponding raise method. Again, the parameters are made available in Intellisense and are checked at compile time.

```

try
{
    Fail12();
}
catch (Exception ex)
{
    CMSEException.Raise_LCS_INVALID_XML (
        ex);
}
static void Fail12()
{
    CMSEException.Raise_LCS_INVALID_XML (
        ex);
}

```

Figure 3.2.3. Intellisense shows exceptions that can be raised and the parameter list

Note that when an exception is raised for each exception resource, the type of the exception class is passed to ConfigException. This actually helps during debugging of the application. When a localized exception is raised, a developer can see in the debug window the type of the exception raised. In this particular example, the type of the exception would be displayed as CMSEException when debugging the application. Refer to the figure below for an example.

The screenshot shows a Visual Studio IDE with a code file on the right and a Watch window at the bottom. The code file contains a try-catch block that catches an exception and writes the message to the console. The Watch window shows the exception object 'ex' of type 'TestConsoleApp.CMSEException'.

Code File:

```

20 try
21 {
22     Fail12();
23 }
24 catch (Exception ex)
25 {
26     Console.WriteLine("Exception message: " + ex.Message);
27 }
28
29 Console.WriteLine("Resource1 = " + ClientModelSchema.MISSING_QUERY());
30
31 static void Fail12()
32 {
33     try
34     {
35

```

Watch Window:

Name	Value	Type
ex	("Error parsing XML. The value 'XYZ' is not valid.")	System.Exception
TestConsoleApp.CMSEException	(TestConsoleApp.CMSEException)	TestConsoleApp.CMSEException
System.Object	(TestConsoleApp.CMSEException)	System.Object
ClassName	null	string
COMPlusExceptionCor	-532459699	int
ExceptionMethod	<undefined value>	System.Reflection.MethodBase
ExceptionMethodStr	null	string
HelpURL	null	string
HRESULT	-2146232832	int
InnerException	("Cannot process the string.")	System.Exception

Figure 3.2.4. Watch window shows type of exception

As you can see, the type CMSEException is shown on the right side of the watch window in debug mode.

Using the exception classes in the code is also relatively simple. A sample code below simulates catching an exception from an application.

```

static void Main(string[] args)
{
    // Set the locale to German Switzerland
    Thread.CurrentThread.CurrentCulture = CultureInfo.CreateSpecificCulture("de-CH");

    try
    {
        Fail();
    }
    catch (Exception ex)
    {
        Console.WriteLine("Exception message = " + ex.ToString());
        Console.ReadLine();
    }

    static void Fail()
    {
        try
        {
            Fail2();
        }
        catch (Exception ex)
        {
            CMSEException.Raise_LCS_INVALID_XML(ex, "XYZ");
        }
    }

    static void Fail2()
    {
        throw new ApplicationException("Cannot process the string.");
    }
}

```

Figure 3.2.5. Sample application that uses the localized exception class

In this sample, the `Fail2` method throws an application exception which is caught by the `Fail` method. The `Fail` method then consumes this exception and raises its own exception (`CMSEException`) passing in a parameter of "XYZ". Finally, the application catches the exception and calls the overridden `ToString` method of the exception to display all exception messages including those from the inner exceptions or the exceptions consumed by `CMSEException`. The example output below shows how the `ToString` method concatenates all the exception messages into one string message.

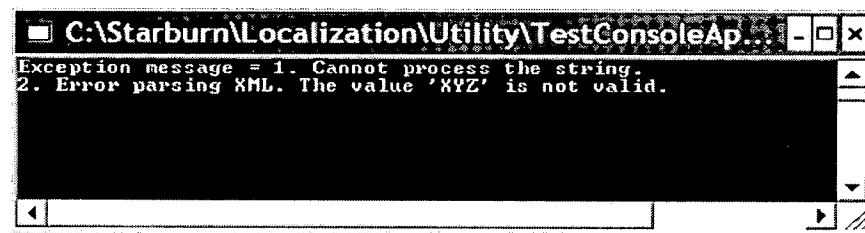


Figure 3.2.6. Sample output showing list of exception messages

You will also notice in the code shown in Figure 3.2.2 that the `CMSEException` class is marked with the `Serializable` attribute. This supports the serialization of the exception classes. This means that exceptions thrown from a different process space can be serialized across the wire and be effectively reconstructed on the receiving end. Thus, exception messages can be passed and displayed across different process spaces.

3.3 DVResourceGenerator

There is still the problem of hand coding the corresponding string and exception resource classes. It turns out that a significant amount of time will be needed by the developer to just create these classes from the resource files. Furthermore, keeping the resource files and the resource classes synchronized is also a time consuming task.

In order to alleviate this problem, a custom tool will be provided to the developers. This custom tool called `DVResourceGenerator` will be attached to the resource files and it should generate the corresponding classes. Another advantage of this tool is that every time a resource file is edited and saved, the resource classes are also regenerated. Hence, all the developer need to do now is just create a resource file (*.resx) and associate it with the custom tool. Any updates (add, edit or delete) done to the resource file should automatically reflect back to the corresponding resource class.

To enable this custom tool, you first need to register it so that Visual Studio will know where to find it. This can be accomplished by opening up a command prompt and issuing a `regasm` command to the custom tool. This is shown in the figure below:

```

C:\WINDOWS\System32\cmd.exe
Setting environment for using Microsoft Visual Studio .NET 2003 tools.
(If you have another version of Visual Studio or Visual C++ installed and wish
to use its tools from the command line, run vcvars32.bat for that version.)

C:\Starburn\Localization>regasm /codebase DVResourceGenerator.dll
Microsoft (R) .NET Framework Assembly Registration Utility 1.1.4322.573
Copyright (C) Microsoft Corporation 1998-2002. All rights reserved.

Regasm warning: Registering an unsigned assembly with /codebase can cause your a
sembly to interfere with other applications that may be installed on the same c
omputer. The /codebase switch is intended to be used only with signed assemblies
. Please give your assembly a strong name and re-register it.
Types registered successfully

C:\Starburn\Localization>

```

Figure 3.3.1. Registering the custom tool

This publishes the tool in the windows registry and allows Visual Studio to use this custom tool.

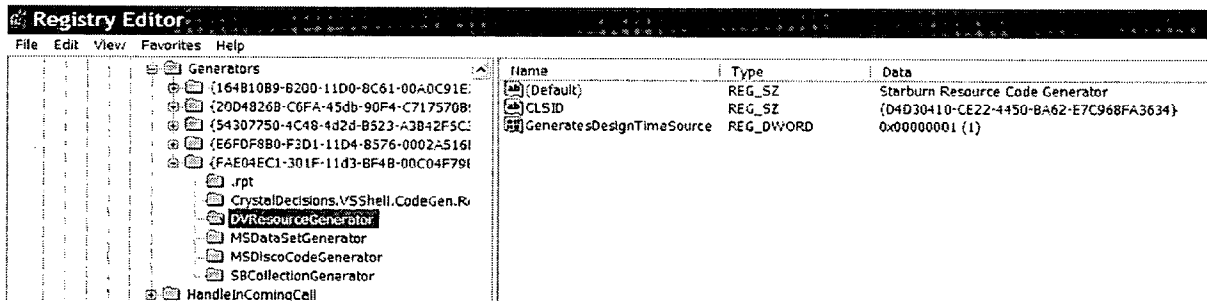



Figure 3.3.2. Publishing the custom tool in the registry

Figure 3.3.3 shows a sample resource file called `ClientModelSchema.resx`. The property window for this resource file shows where a **Custom Tool** can be specified. In this particular case, what we want to do is to create the corresponding resource class called `ClientModelSchema.cs` which will contain the methods needed to access the localized resource strings. For the developer, after creating the resource file and adding all the resources, it is simply a matter of typing the custom tool name in the property window. Once the custom tool `DVResourceGenerator` has been specified (refer to Figure 3.3.4), the corresponding resource class should be generated. By default, Visual Studio does not display the generated class. In order to see the class that has been generated and added to the application, click on the **Show All** icon in the Solution Explorer. A  symbol will be displayed next to the resource file. Click on the symbol to expand the view and the generated class should be displayed. Double-clicking on the class will open up the generated class in the Visual Studio workspace (refer to Figure 3.3.4).

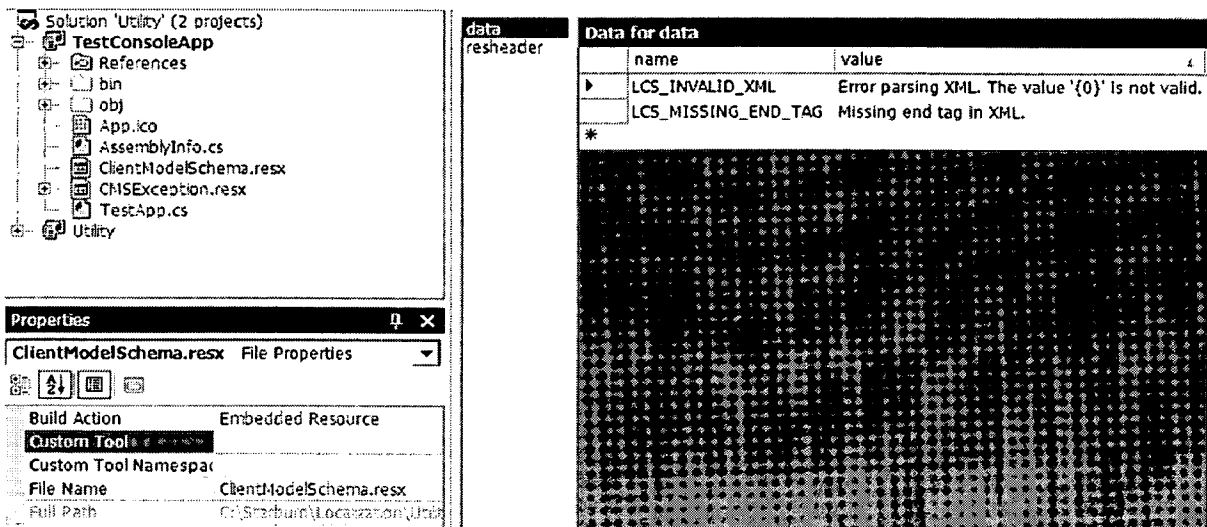


Figure 3.3.3. Sample resource file

You will notice from the generated class a comment that states that this class has been auto-generated and that directly editing the class is not advisable because all changes will be overwritten when the class is regenerated.

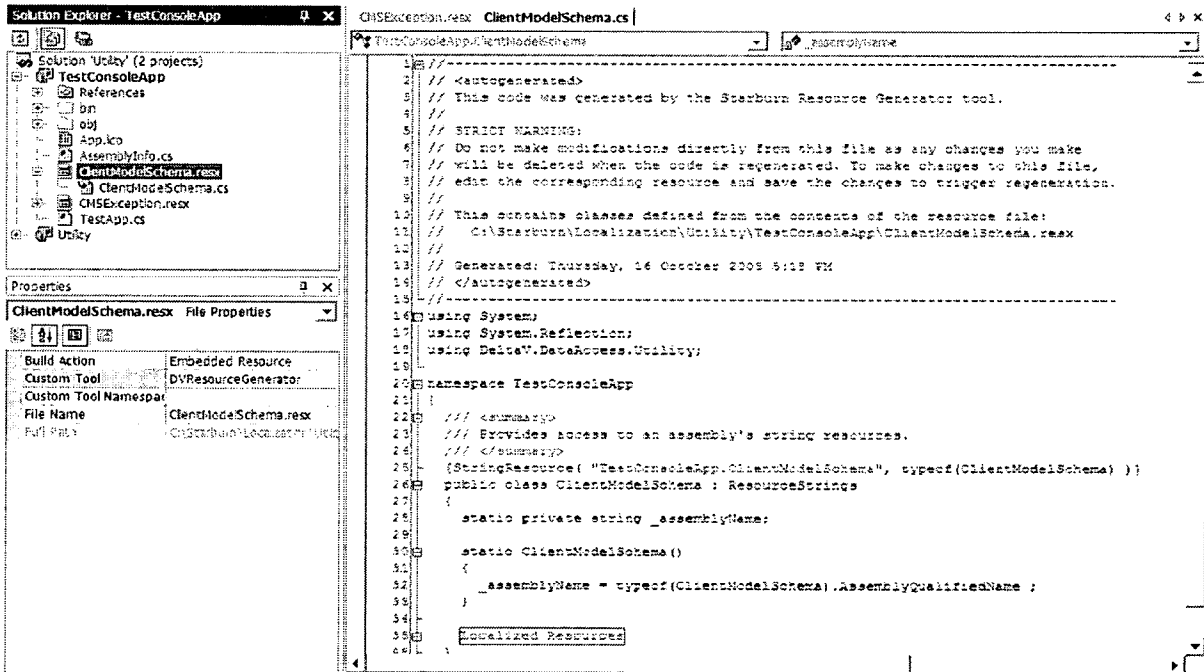


Figure 3.3.4. The generated resource class

3.4 Summary

The framework provided by the globalization classes (ResourceStrings, StringResourceAttribute, and ConfigException) provides the foundation by which an application in Starburn can be globalized. And as an additional help for developers, the use of this framework is made more intuitive and easy by the use of the custom tool DVResourceGenerator. This custom tool hides the complexity of managing resource classes and resources in general. All that a developer has to be concerned with is identifying the resources to be localized and where in their application code these localized resources need to be accessed.

The framework combined with the custom tool addresses the first objective of this prototype. Localization can now be incorporated in Starburn applications with minimal impact and effort on the part of the developer.

4 Localization Support

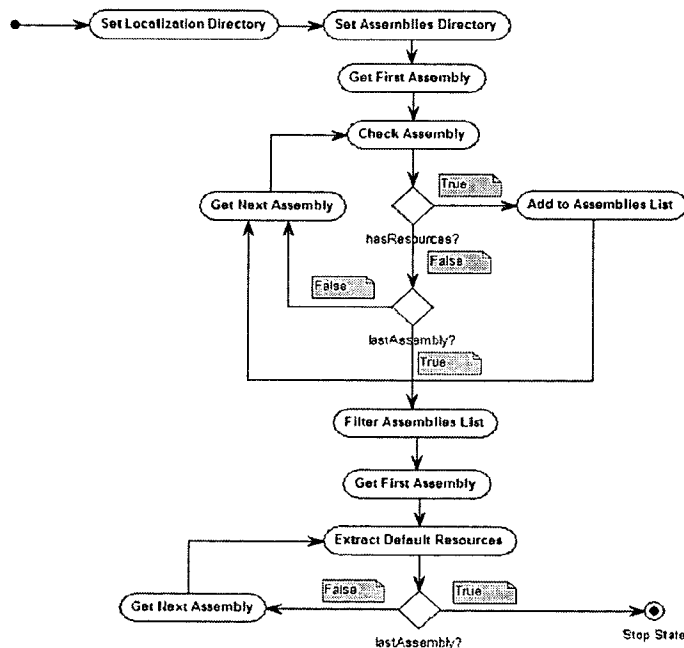
Once the application has been globalized and is deemed ready for localization, a resource toolkit needs to be in place to help with the actual localization of the application. The first step towards this toolkit is coming up with a resource tool that can analyze the .NET assemblies and create the corresponding satellite assemblies from the translated resources.

The resource tool should be able to do the following:

- 1 Read an application assembly and extract the default localizable resources.
- 2 Provide a translation editor for translating the default resources.
- 3 Read the translated resources and create the localized satellite assembly.

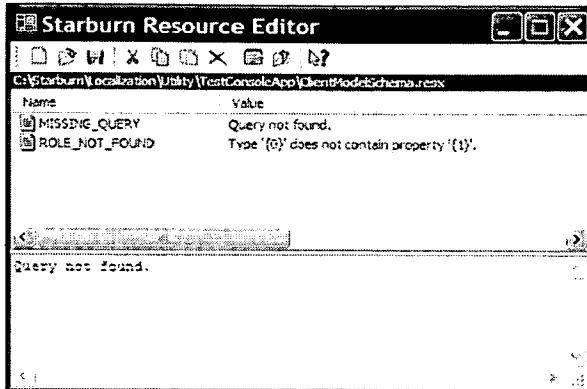
4.1 Extracting the Default Resources

The activity diagram below shows how the prototype extracts the resources from the given assemblies:



4.2 Translating the Resources

An editor similar to the mock-up screen below will be used to translate the default resources to a specific culture.



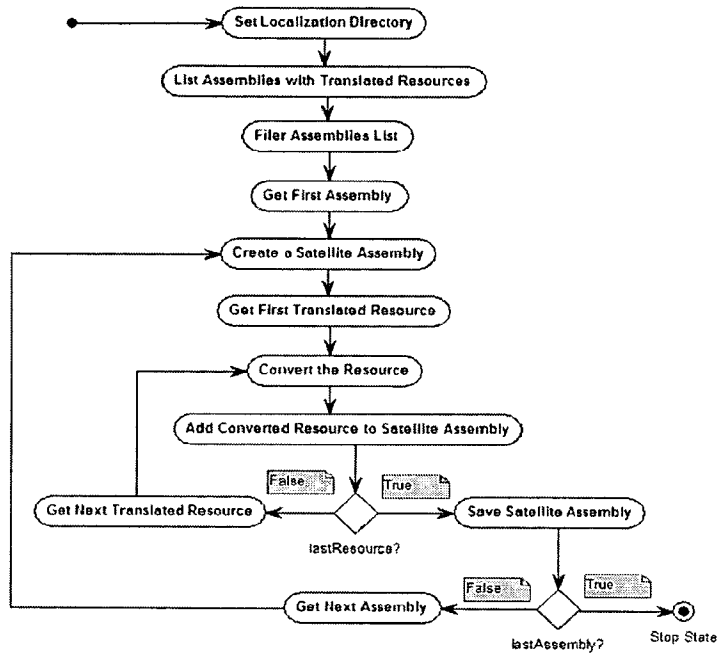
When all the resources have been translated, clicking on the save icon should bring up a window similar to the one below to indicate which culture the translation will be saved to. The correct culture name will be appended to the filename of the translated resource.



For example, when the default resource `ClientModelSchema.resx` gets translated into the neutral culture French, it will be saved as `ClientModelSchema.fr.resx`. If a specific culture is chosen like French-Canada, the resource file will be saved as `ClientModelSchema.fr-CA.resx`.

4.3 Creating the Localized Satellite Assemblies

The activity diagram below shows how the prototype creates the satellite assemblies:



The satellite assemblies created will automatically be placed in the corresponding culture folder under the main assemblies folder. This is in line with the resource fallback scheme implemented in the .NET Framework.

5 Appendix – Globalization Classes

5.1 ResourceStrings Class

```
using System;
using System.Resources;
using System.Reflection;
using System.Globalization;
using System.Threading;
using System.Collections;

namespace DeltaV.DataAccess.Utility
{
    public class ResourceStrings
    {
        #region Public GetString Method (Overloaded)
        /// <summary>
        /// Loads an unformatted string
        /// </summary>
        /// <param name="resourceId">Identifier of string resource</param>
        /// <returns>string</returns>
        public static string GetString( string typeName, string resourceId )
        {
            string resourceValue = null;

            ResourceManager rm = GetResourceManager( typeName );

            if ( rm != null )
            {
                resourceValue = rm.GetString( resourceId );
            }

            if ( resourceValue == null )
            {
                resourceValue = resourceId;
            }

            return resourceValue;
        }

        /// <summary>
        /// Loads a formatted string
        /// </summary>
        /// <param name="resourceId">Identifier of string resource</param>
        /// <param name="objects">Array of objects to be passed to string.Format</param>
        /// <returns>string</returns>
        public static string GetString( string typeName, string resourceId, params object[]
objects )
        {
            string resourceValue = null;

            ResourceManager rm = GetResourceManager( typeName );

            if ( rm != null )
            {
                resourceValue = rm.GetString( resourceId );
            }

            if ( resourceValue != null )
            {
                return string.Format( resourceValue, objects );
            }
            else
            {
                return resourceId;
            }
        }
    }
    #endregion

    #region Resource Managers code
    private static Hashtable _ResourceManagers = null ;

    public static ResourceManager RegisterResourceDll ( string typeName,
string baseName, Assembly assembly )
```

```

    {
        ResourceManager rm = new ResourceManager ( baseName, assembly ) ;
        ResourceManagers.Add ( typeName, rm ) ;
        return rm ;
    }

    private static Hashtable ResourceManagers
    {
        get
        {
            lock ( typeof ( ResourceStrings ) )
            {
                if ( _ResourceManagers == null )
                {
                    _ResourceManagers = new Hashtable();
                }
            }
            return _ResourceManagers ;
        }
    }

    private static ResourceManager GetResourceManager ( string typeName )
    {
        ResourceManager rm = ResourceManagers[typeName] as ResourceManager ;
        if ( rm == null )
        {
            object[] stringResources = Type.GetType ( typeName ).GetCustomAttributes (
                typeof ( StringResourceAttribute ), true ) ;
            if ( stringResources != null )
            {
                StringResourceAttribute stringResource = stringResources[0] as
StringResourceAttribute;
                rm = RegisterResourceDll (
                    typeName,
                    stringResource.baseResource,
                    stringResource.assembly ) ;
            }
        }
        return rm ;
    }
}
#endregion
}
}

```

5.2 StringResourceAttribute Class

```

using System;
using System.Reflection;

namespace DeltaV.DataAccess.Utility
{
    [AttributeUsage(AttributeTargets.Class)]
    public class StringResourceAttribute : Attribute
    {
        public readonly string baseResource ;
        public readonly Assembly assembly ;
        public StringResourceAttribute ( string baseResourceName, Type t )
        {
            this.baseResource = baseResourceName ;
            this.assembly = t.Assembly ;
        }
    }
}

```

5.3 ConfigException Class

```

using System;
using System.Collections;
using System.Text;
using System.Runtime.Serialization;

namespace DeltaV.DataAccess.Utility
{
    [Serializable]
    public class ConfigException : ApplicationException
    {
    }
}

```

```

{
    #region Constructors
    public ConfigException () {}
    public ConfigException ( string exceptionMessage ) : base ( exceptionMessage ) {}
    public ConfigException ( string exceptionMessage, Exception exception ) :
        base(exceptionMessage, exception) {}
    public ConfigException ( SerializationInfo info, StreamingContext context ) :
        base ( info, context ) {}
    #endregion

    #region Overloaded method for raising the exception
    public static void Raise ( Type exceptionType, string exceptionMessage )
    {
        throw Activator.CreateInstance (
            exceptionType,
            new Object[] { exceptionMessage } ) as ConfigException ;
    }

    public static void Raise ( Type exceptionType, string exceptionMessage, Exception
exception )
    {
        throw Activator.CreateInstance (
            exceptionType,
            new Object[] { exceptionMessage, exception } ) as ConfigException ;
    }
    #endregion

    #region Overridden Methods of ApplicationExtension
    /// <summary>
    /// This creates a list of exception messages composed of the original exceptions
    /// as well as the inner exceptions.
    /// </summary>
    /// <returns></returns>
    public override string ToString()
    {
        ArrayList messageList = new ArrayList() ;

        Exception ex = this ;
        while ( ex != null )
        {
            messageList.Insert ( 0, ex.Message ) ;
            ex = ex.InnerException ;
        }

        int idx = 1 ;
        StringBuilder sb = new StringBuilder() ;
        foreach ( string s in messageList )
        {
            sb.AppendFormat ( "{0}. {1}\n", idx++, s ) ;
        }

        return sb.ToString () ;
    }
    #endregion
}
}

```

Diagram Control

Table of Contents:

1	INTRODUCTION	325
1.1	OVERVIEW	325
1.2	DEFINITIONS.....	325
1.3	REFERENCES	325
1.4	DELIVERABLES	325
2	OBJECTIVES.....	325
3	USER INTERFACE.....	327
4	OBJECT MODEL	328
4.1	USE CASES.....	329
4.1.1	User launches the function block diagram view	330
4.1.2	User opens an existing module in diagram control for editing.....	331
4.1.3	User adds a connection between two connectors	332
4.1.4	User adds a new function block to the module	333
4.1.5	User adds a new parameter to a function block	334
4.1.6	User deletes a function block.....	335
4.1.7	User saves the module	335
4.1.8	User right clicks on the diagram control to view the properties of MOD1.....	336
4.1.9	User opens an existing module in diagram control for debugging	337
4.1.10	User embeds the diagram control in a runtime environment.....	337
5	INTERFACE DESIGN FOR DIAGRAM CONTROL	338
5.1	IDIAGRAMCONTROL.....	338
5.1.1	IDiagramControl.createDiagram	338
5.1.2	IDiagramControl.addFigure.....	338
5.1.3	IDiagramControl.addConnection	339
5.1.4	IDiagramControl.deleteFigure.....	340
5.1.5	IDiagramControl.deleteConnection.....	340
5.1.6	IDiagramControl.updateFigure	341
5.1.7	IDiagramControl.getFigureInfo	341
5.1.8	IDiagramControl.draw	342
5.1.9	IDiagramControl.createFigureInfo	342
5.1.10	IDiagramControl.createShapeInfo	343
5.1.11	IDiagramControl.createConnectorInfo	343
5.1.12	IDiagramControl.setState.....	344
5.1.13	IDiagramControl.setDiagramControlProperties	345
6	EVENTS	346
6.1.1	NewFigure.....	346
6.1.2	NewConnection.....	346
6.1.3	VerifyConnection.....	346
6.1.4	DeleteFigure	347
6.1.5	DeleteConnection	347
6.1.6	GetFigureContextMenu.....	347
6.1.7	GetDiagramContextMenu	348
6.1.8	NewPosition	348
6.1.9	NewSize	348
7	DATASETS	349
7.1	CFIGUREINFO	349

7.2	CShapeInfo	349
7.3	CShape_LineInfo.....	350
7.4	CShape_RectangleInfo	350
7.5	CShape_EllipseInfo	350
7.6	CShape_CrossInfo.....	350
7.7	CShape_TerminatorInfo.....	350
7.8	CShapeStyle	350
7.9	CConnectorInfo.....	351
7.10	CConnectionInfo	351
7.11	CDiagramControlProperty	351
7.12	CDiagramInfo	351
7.13	CConstraintInfo.....	352

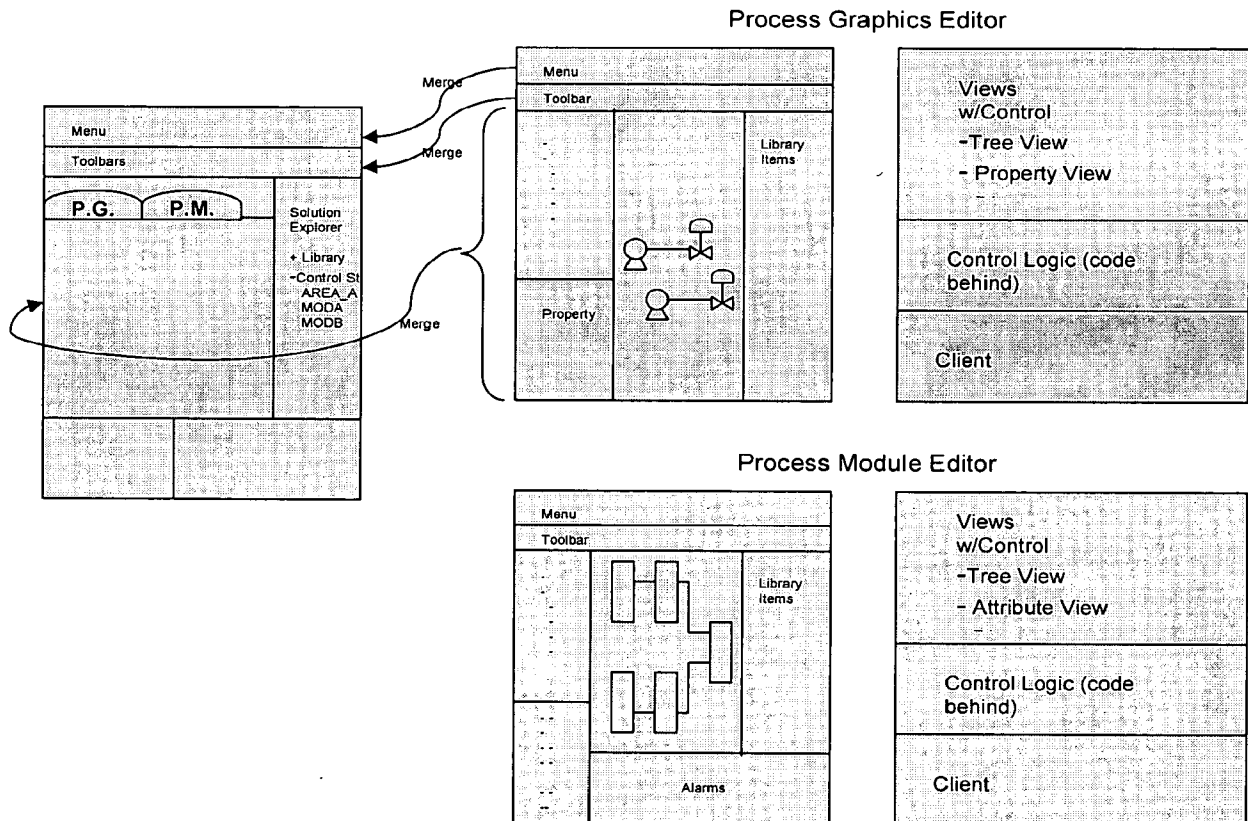
Table of Figures:

Figure 1 - Configuration WorkSpace	327
Figure 2 - Architecture of View and Diagram Control.....	328
Figure 3 - Data Collaboration Diagram for Open an Existing Module	331
Figure 4 - Data Collaboration Diagram for Add a New Connection	332
Figure 5 - Data Collaboration Diagram for Add a New Function Block	333
Figure 6 - Data Collaboration Diagram for Add a New Parameter	334
Figure 7 - Data Collaboration Diagram for Save a Module	335
Figure 8 - Data Collaboration Diagram for Debug	337

1 Introduction

1.1 Overview

There are several graphical editors in current DeltaV system to help customers create and debug control strategies in an intuitive way. The graphical editors include Control Studio Diagram Control (edit and online mode), Recipe Studio Diagram Control, and Operator Interface graphic editor. Control Studio Diagram Control and Recipe Studio Diagram Control share the same code set, but draw different shapes based on the context. Operator Interface graphic editor uses completely different drawing tool.



1.2 Definitions

1.3 References

1.4 Deliverables

2 Objectives

The objective for this prototype is to build a generic diagram control that can be used by various graphic editors includes Control Studio, Recipe Studio, DeltaV Operator Interface. The diagram controls should also be able to be embedded in run time applications in read-only mode.

In order for the diagram control to be used in many ways, it is critical that the diagram control is independent of any specific context information. It is up to the containing application (views or applets) to provide the correct data for the diagram control to function.

Here is a list of objectives:

1. The diagram control is contained by a view.
2. The view is contained by a package.
3. The diagram control does not talk to Client Model directly. It will communicate with the Client Model via its containing view.
4. The diagram control gets pre-defined dataset from its containing view.
5. The diagram control should be able to change the drawing without having to refresh the entire diagram.
6. When user interacts with the diagram control, the diagram control will perform the operations and fire events to the containing view.
7. The diagram control stores the object related state information (for instance, font setting, color setting, etc) to the configuration database via its containing view.
8. The diagram control will store its global state information (size, maximized or minimized) in Registry.
9. The diagram control makes no distinction of a FB module, a SFC, or a display. The constraints (for example, which connectors are allowed to be connected) are captured by the dataset passed to the diagram control by the containing view.
10. The diagram control gets its context menu from the containing view. When user invokes a context menu, the diagram control will perform some operation and then fire the event back to the containing view. The context menu may include Save, Copy and Paste, Delete, Undo, Redo etc.
11. The containing view will act as a data store for the diagram control. The diagram control will not cache any interim data.
12. The diagram control will publish a set of interfaces for the containing view to tie in.

3 User Interface

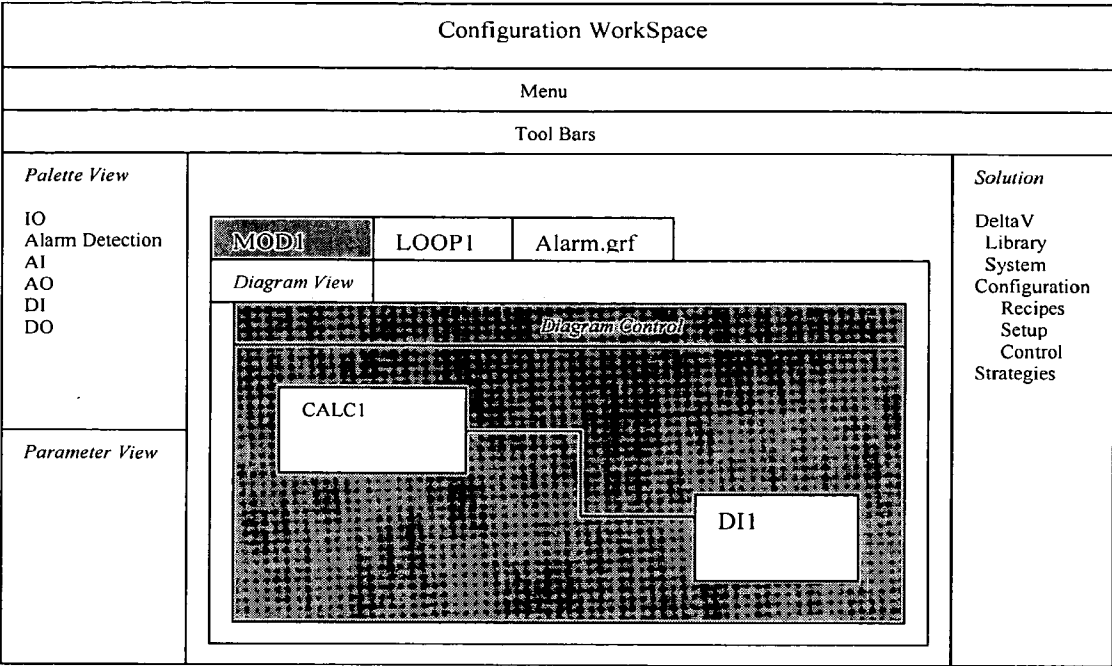


Figure 1 - Configuration WorkSpace

4 Object Model

The object model illustrates how the diagram control is related to the configuration workspace and client model.

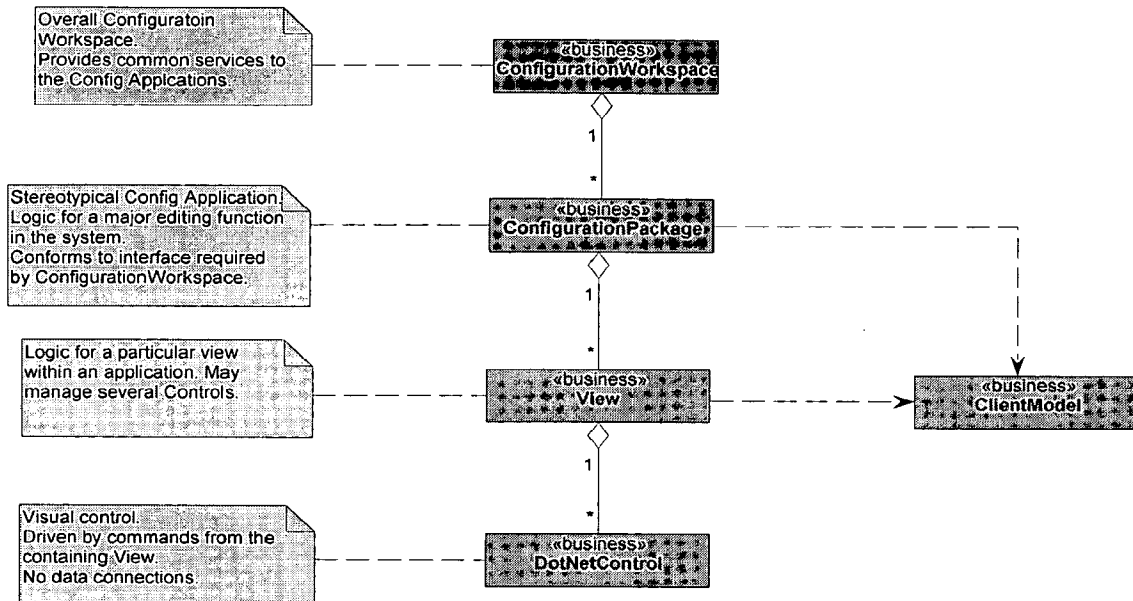
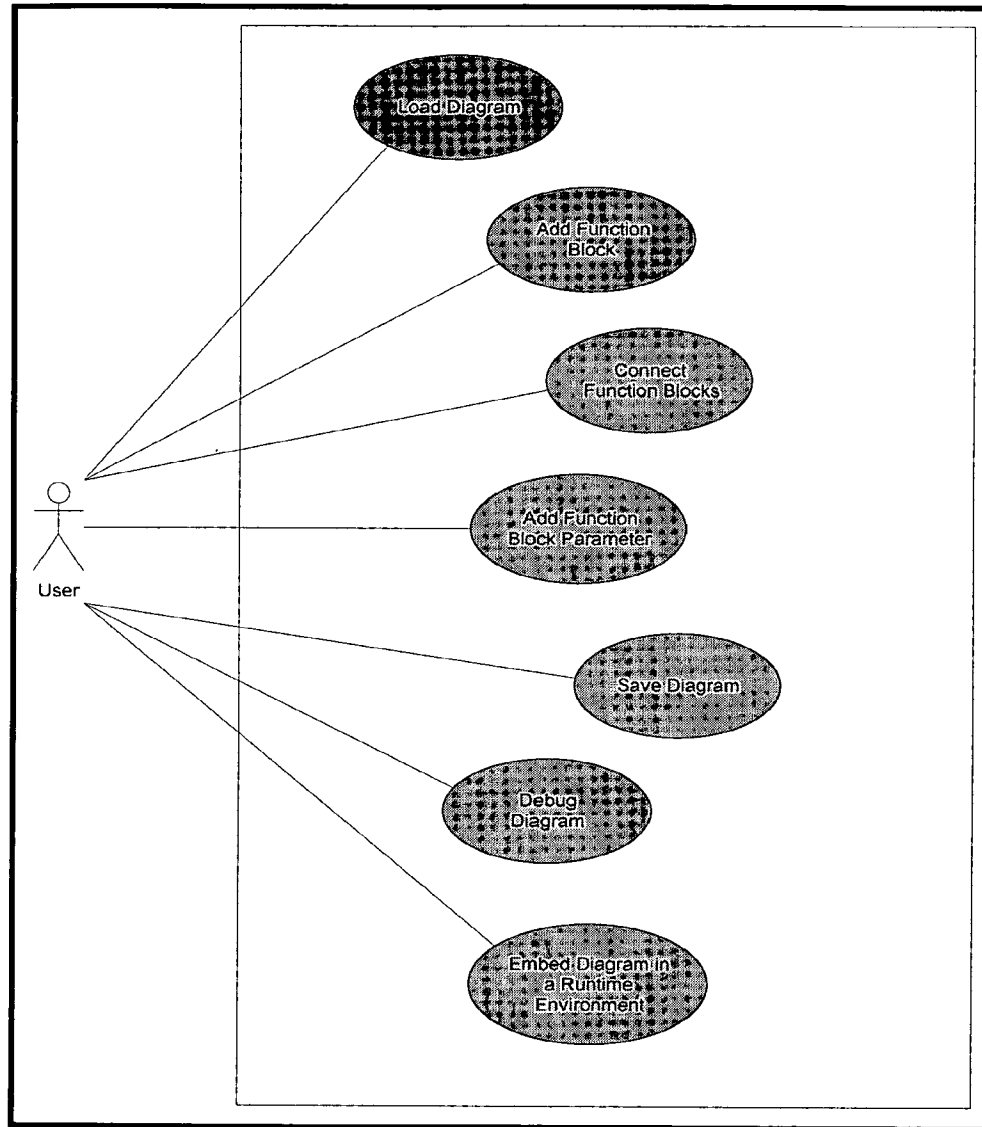


Figure 2 - Architecture of WorkSpace

4.1 Use Cases

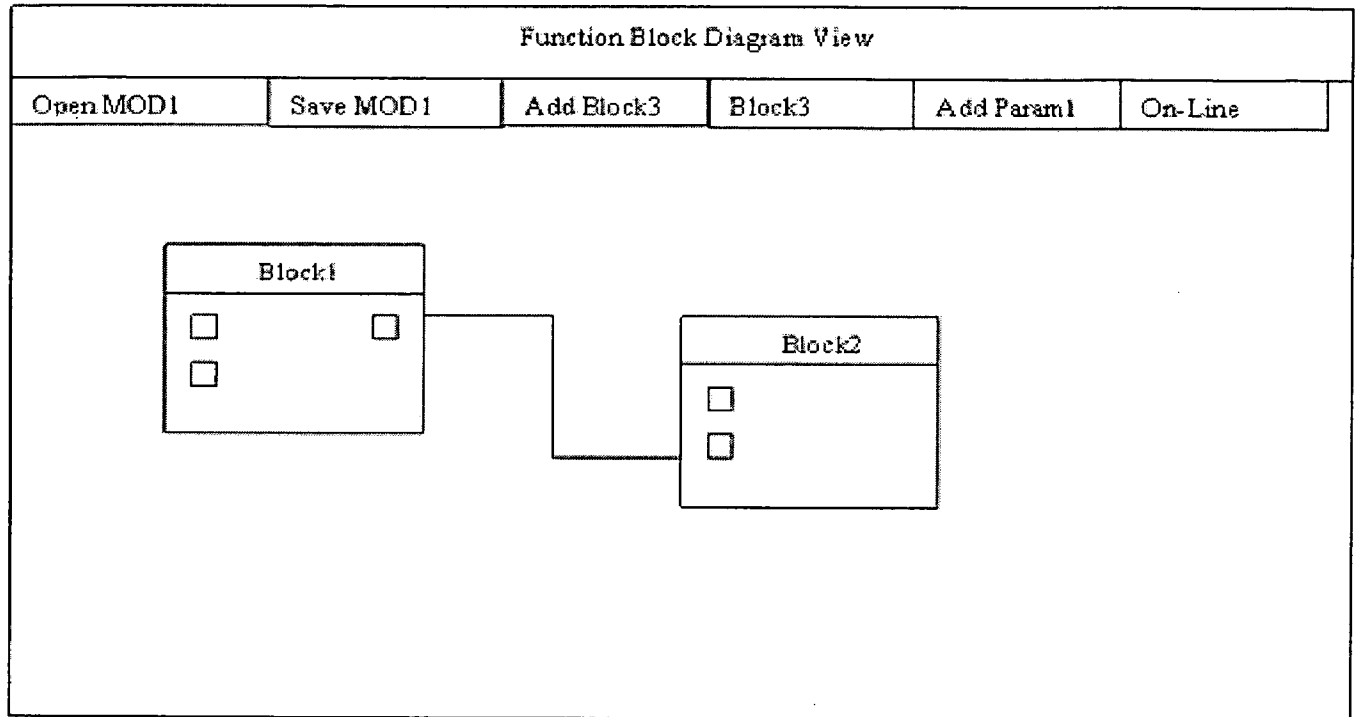
All use cases will operate within a containing view that generates the correct data for the diagram control.



4.1.1 User launches the function block diagram view

The diagram view is a standalone application. User can launch it from the “Run” command.

System Response: The application will launch with a function block view that contains a diagram control.



4.1.2 User opens an existing module in diagram control for editing

4.1.2.1 Preconditions

- a) Diagram Control is empty.

4.1.2.2 User Actions/System Response

User Actions	System Response
User selects the "Open MOD1" button from the container.	The diagram view is loaded with all the objects positioned with connection lines based upon the last time saved.

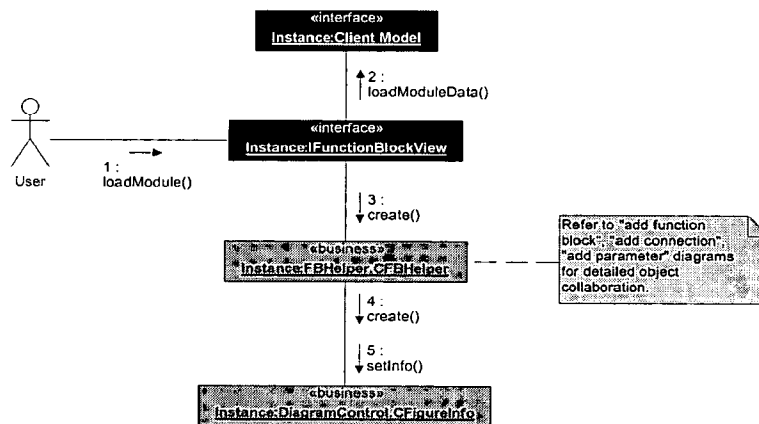


Figure 3 - Data Collaboration Diagram for Open an Existing Module

4.1.3 User adds a connection between two connectors

4.1.3.1 Preconditions

- Diagram view is launched. MOD1 is loaded.
- At least one unused input and an output connection may exist on the diagram.

4.1.3.2 User Actions/System Response

User Actions	System Response
User positions mouse cursor on an input knob or an available output knob.	The connection cursor is displayed.
User depresses left mouse button.	
User drags cursor to knob of opposing type.	A transitory line is displayed and updated as the user drags the mouse. The target hit cursor is displayed when positioned over a valid connection location.
User releases left mouse button.	If the user releases on a valid opposing connection an appropriate line is displayed. If the user does not release on a valid opposing connection the transitory line disappears.

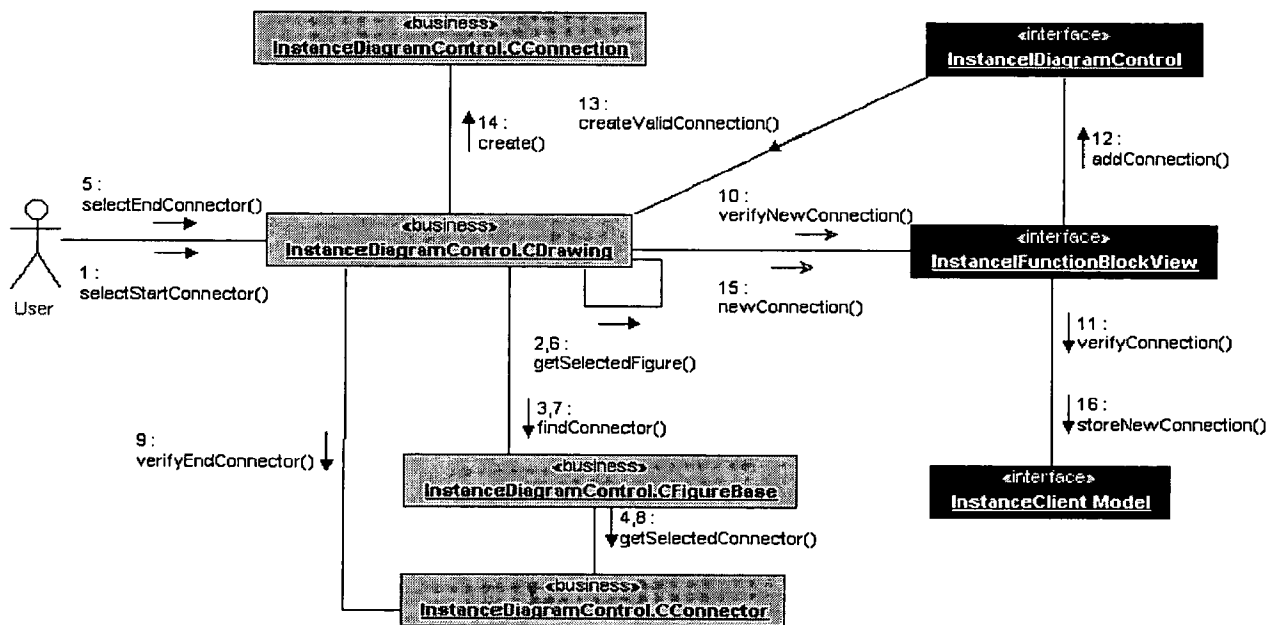


Figure 4 - Data Collaboration Diagram for Add a New Connection

4.1.4 User adds a new function block to the module

4.1.4.1 Preconditions

a) Diagram view is launched, MOD1 is loaded.

4.1.4.2 User Actions/System Response

User Actions	System Response
User clicks on "Add Block3" button	Block3 appears in the diagram control in a pre-specified position.

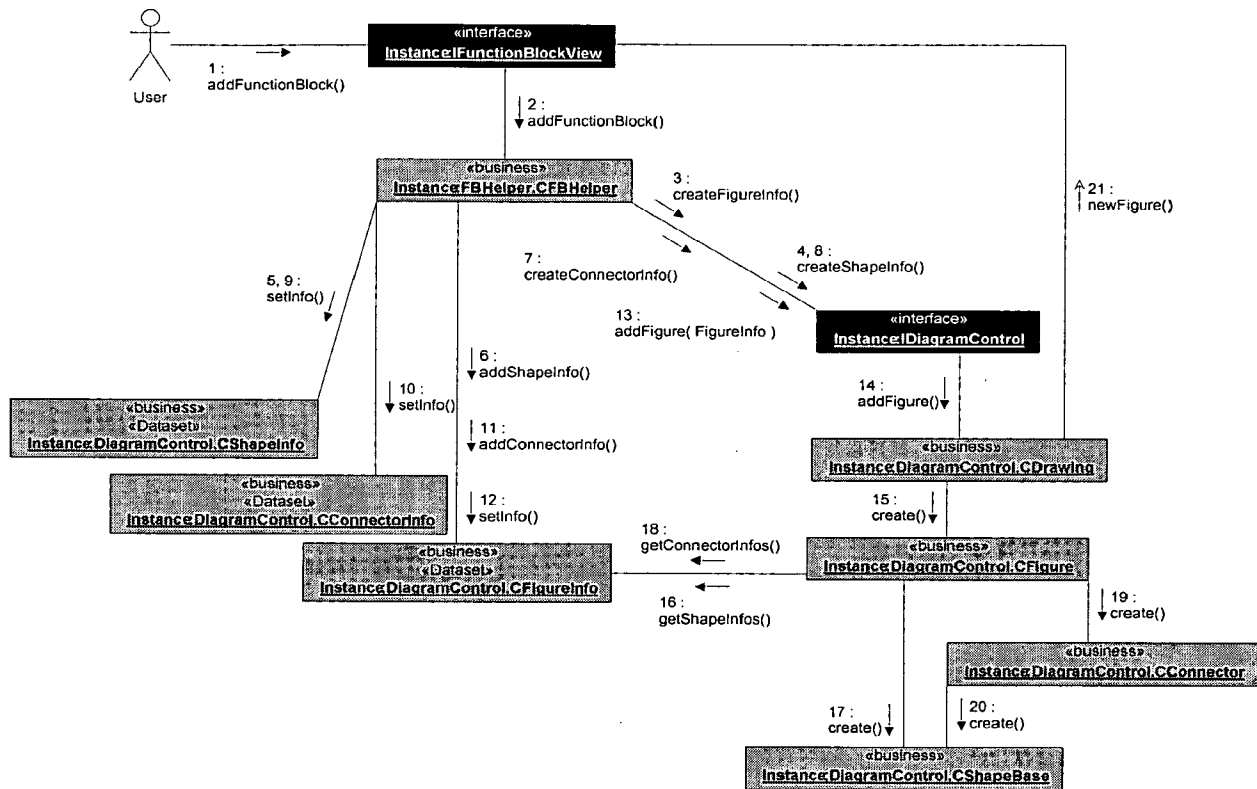


Figure 5 - Data Collaboration Diagram for Add a New Function Block

4.1.5 User adds a new parameter to a function block

4.1.5.1 Preconditions

a) Diagram view is launched and MOD1 is loaded.

4.1.5.2 User Actions/System Response

User Actions	System Response
User clicks on "Add Param1" button	Param1 appears in the diagram control in a pre-specified position.

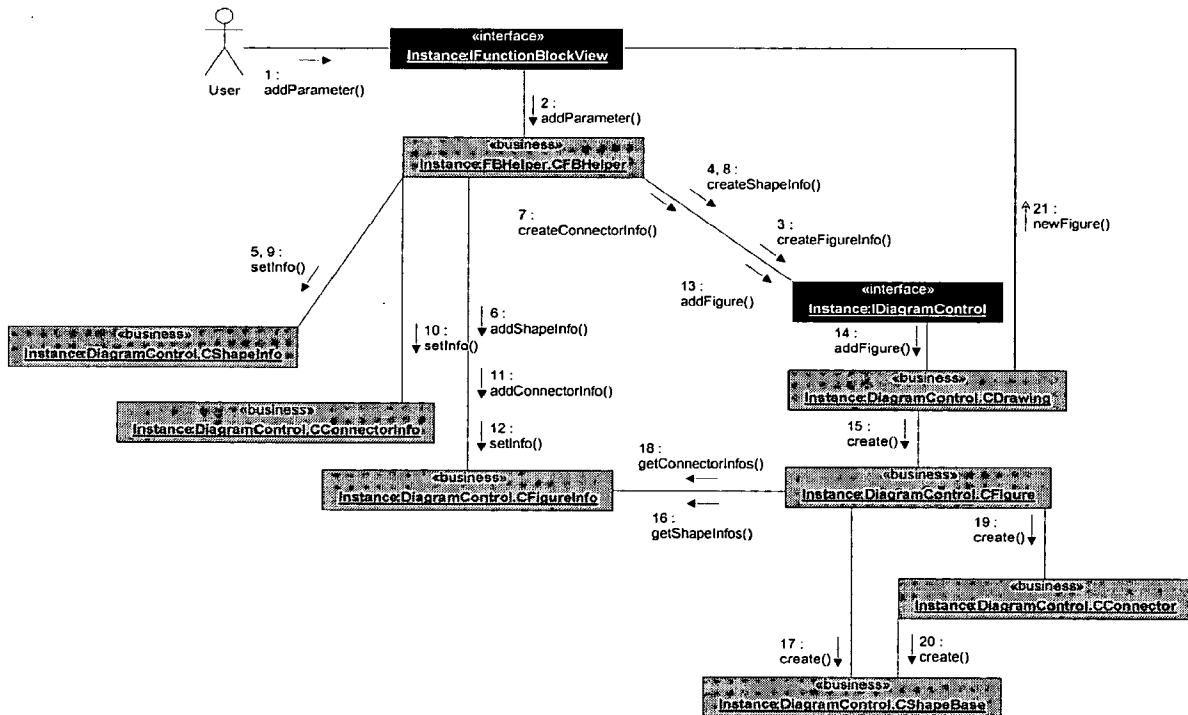


Figure 6 - Data Collaboration Diagram for Add a New Parameter

4.1.6 User deletes a function block

4.1.6.1 Preconditions

- a) Diagram view is launched and MOD is loaded.

4.1.6.2 User Actions/System Response

User Actions	System Response
User selects a function block "Block1".	Block1 is selected
User clicks on the "Delete" button	Block1 is deleted if the deletion is allowed, otherwise, an error message will be displayed

4.1.7 User saves the module

4.1.7.1 Preconditions

- a) Diagram view is launched and MOD is loaded.

4.1.7.2 User Actions/System Response

User Actions	System Response
User clicks on "Save MOD1" button.	MOD1 is saved back to the Client Model and the objectivity database.

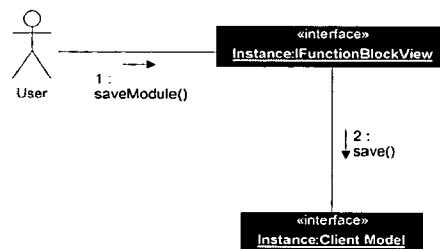


Figure 7 - Data Collaboration Diagram for Save a Module

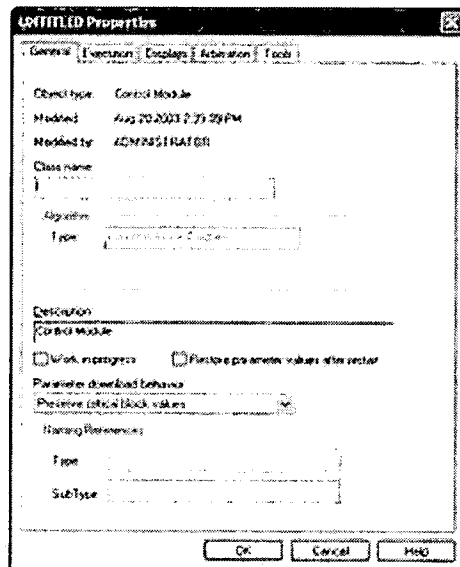
4.1.8 User right clicks on the diagram control to view the properties of MOD1

4.1.8.1 Preconditions

- a) Diagram view is launched and MOD is loaded.

4.1.8.2 User Actions/System Response

User Actions	System Response
User right clicks on an empty space of the diagram view.	A context menu pops up with one option "Properties"
User selects on "Properties"	A dialog pops up display the properties of MOD1



4.1.9 User opens an existing module in diagram control for debugging

4.1.9.1 Preconditions

- a) Diagram view is launched and MOD1 is loaded.

4.1.9.2 User Actions/System Response

User Actions	System Response
User clicks on "On-Line" button.	Diagram view will change the data source from Client Model to read ¹ run time data.

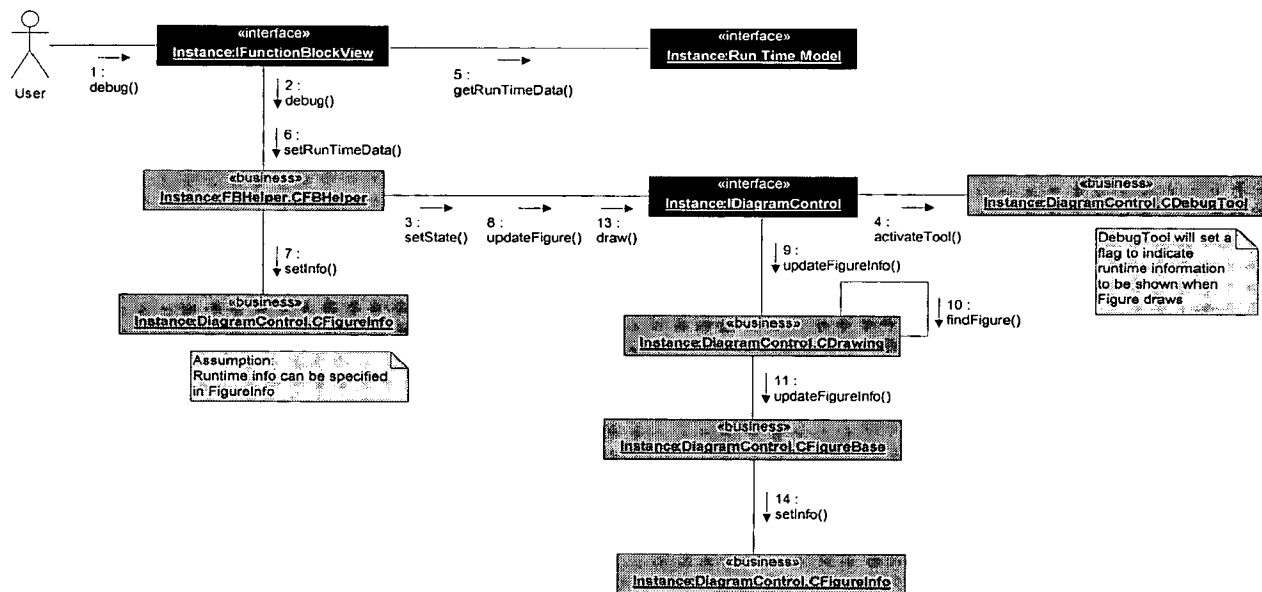


Figure 8 - Data Collaboration Diagram for Debug

4.1.10 User embeds the diagram control in a runtime environment

TBD.

This will require the containing view be self-sufficient? The containing view should support a mode that is not dependant on the workspace frame work. Or should we just embed an instance of the workspace with a very restrictive mode (open one module read only)?

¹ Depending on the progress of this project, the run time data may be a set of pre-specified value.

5 Interface design for Diagram Control

5.1 IDiagramControl

5.1.1 IDiagramControl.createDiagram

Method

```
createDiagram(  
    CDiagramInfo diagramInfo  
);
```

Description

This interface method is called by the containing application to pass in a pre-defined dataset describing the diagram to draw to the diagram control. This dataset contains the description of the diagram in a way that is understood by the diagram control.

Parameters

Name	Type	Description
diagramInfo	CDiagramInfo	The pre-defined dataset to describe the shapes to draw from the diagram control

Exceptions

Name	Description
DC_FAIL_TO_CREATE_DIAGRAM	This exception will be thrown when the diagram control fails to create the diagram requested.

Comments

This method should be called before any other requests to the diagram control to draw a diagram.

5.1.2 IDiagramControl.addFigure

Method

```
addFigure(  
    String strFigureName,  
    CFigureInfo figureInfo  
);
```

Description

This interface method is called by the containing application to pass in a pre-defined dataset describing the figure to add to the diagram control. This dataset contains the description of the objects in a way that is understood by the diagram control.

Parameters

Name	Type	Description
strFigureName	String	The name of the figure being added
figureInfo	CFigureInfo	The pre-defined dataset to describe the shapes to draw

		from the diagram control
--	--	--------------------------

Exceptions

Name	Description
DC_FAIL_TO_ADD_FIGURE	This exception will be thrown when the diagram control fails to add the figure requested.

Comments

Depending on what shapes are contained in the **CFigureInfo** object, the diagram control will create corresponding shape objects accordingly. The shape objects will contain its corresponding data.

5.1.3 IDiagramControl.addConnection

Method

```
addConnection(  
    CConnectionInfo connectionInfo  
);
```

Description

This interface method is called by the containing application to pass in a pre-defined dataset of connection to add to the diagram control. This dataset contains the description of the objects in a way that is understood by the diagram control.

Parameters

Name	Type	Description
connectionInfo	CConnectionInfo	The pre-defined dataset to describe the connection to draw from the diagram control

Exceptions

Name	Description
DC_FAIL_TO_ADD_CONNECTION	This exception will be raised when the connectors associated to this connection are not created in the diagram control.

Comments

This method assumes that the associated connectors have been created. If this condition is not satisfied, exception will be raised.

5.1.4 IDiagramControl.deleteFigure

Method

```
deleteFigure(  
    String strFigureName  
);
```

Description

This interface method is called by the containing application to ask the diagram control to remove a figure based on the figure name passed.

Parameters

Name	Type	Description
strFigureName	String	The name of the figure to be deleted

Exceptions

Name	Description
DC_FAIL_TO_DELETE_FIGURE	This exception will be thrown when the diagram control fails to delete the figure requested.

Comments

This method will remove the figure in the diagram control as well as the connections related to it.

5.1.5 IDiagramControl.deleteConnection

Method

```
deleteConnection(  
    String strStartFigureName,  
    String strStartConnectorName,  
    String strEndFigureName,  
    String strEndConnectorName  
);
```

Description

This interface method is called by the containing application to remove an existing connection between the specified start and end connector names.

Parameters

Name	Type	Description
strStartFigureName	String	The name of the start figure for the connection
strStartConnectorName	String	The name of the start connector for the connection
strEndFigureName	String	The name of the end figure for the connection
strEndConnectorName	String	The name of the end connector for the connection

Exceptions

Name	Description
DC_FAIL_TO_DELETE_CONNECTION	This exception will be raised if the connection can not be located or deleted.

5.1.6 IDiagramControl.updateFigure

Method

```
updateFigure(  
    String strFigureName,  
    CFigureInfo figureInfo  
);
```

Description

This interface method is called by the containing application to ask the diagram control to update and draw the corresponding figure based on the **CFigureInfo** passed.

Parameters

Name	Type	Description
strFigureName	String	Name of the figure to update
figureInfo	CFigureInfo	The pre-defined dataset to describe the shapes to draw from the diagram control

Exceptions

Exception	Description
DC_FAIL_TO_UPDATE_FIGURE	This exception will be thrown when the diagram control fails to update the figure requested.

Comments

This method will update the figure object in the diagram control without having to repaint the entire diagram.

5.1.7 IDiagramControl.getFigureInfo

Method

```
CFigureInfo getFigureInfo(  
    String strFigureName  
);
```

Description

This interface method is called by the containing application to retrieve the **CFigureInfo** object from the diagram control.

Parameters

Name	Type	Description
strFigureName	String	Name of the figure whose info is to be retrieved

Return

Type	Description
CFigureInfo	The pre-defined dataset for figure info

Exceptions

Name	Description
DC_FAIL_TO_GET_FIGUREINFO	This exception will be thrown when the diagram control fails to retrieve the CFigureInfo object.

Comments

This method assumes that the figure requested exists. Otherwise, an exception will be thrown.

5.1.8 IDiagramControl.draw

Method

```
draw();
```

Description

This interface method is called by the containing application to ask the diagram control to draw corresponding figure(s) based on the **CFigureInfo(s)** passed in previously.

Exceptions

Name	Description
DC_FAIL_TO_DRAW	This application exception will be thrown when the diagram control fails to draw.

Comments

This method should be called after methods that add objects to the control (i.e. **AddFigure()**) are called. The user can call multiple add object methods before invoking this method.

5.1.9 IDiagramControl.createFigureInfo

Method

```
CFigureInfo createFigureInfo();
```

Description

This interface method is called by the containing application to create an empty **CFigureInfo** object.

Return

Type	Description
CFigureInfo	The pre-defined dataset for figure info

Exceptions

Name	Description
DC_FAIL_TO_CREATRE_FIGUREINFO	This exception will be thrown when the diagram control fails to create a CFigureInfo object.

Comments

This method will return an empty **CFigureInfo** object to the caller.

5.1.10 IDiagramControl.createShapeInfo

Method

```
CShapeInfo createShapeInfo(SHAPE_TYPE enShapeType);
```

Description

This interface method is called by the containing application to create an empty **CShapeInfo** object.

Parameters

Name	Type	Description
enShapeType	SHAPE_TYPE	The type of shape to create

Return

Type	Description
CShapeInfo	The pre-defined dataset for shape info

Exceptions

Name	Description
DC_FAIL_TO_CREATRE_SHAPEINFO	This exception will be thrown when the diagram control fails to create a CShapeInfo object.

Comments

This method will return an empty **CShapeInfo** object to the caller.

5.1.11 IDiagramControl.createConnectorInfo

Method

```
CConnectorInfo createConnectorInfo();
```

Description

This interface method is called by the containing application to create an empty **CConnectorInfo** object.

Return

Type	Description
CConnectorInfo	The pre-defined dataset for connector info

Exceptions

Name	Description
DC_FAIL_TO_CREATRE_CONNECTORINFO	This exception will be thrown when the diagram control fails to create a CConnectorInfo object.

Comments

This method will return an empty **CConnectorInfo** object to the caller.

5.1.12 IDiagramControl.setState**Method**

```
setState(  
    DiagramState enState  
);
```

Description

This interface method is called by the containing application to set the correct mode.

Parameters

Name	Type	Description
enState	DiagramState	Different modes of the diagram control. The available modes are: EDIT DEBUG ONLINE READONLY The default mode is EDIT.

Exception

Exception	Description
DC_FAIL_TO_SET_STATE	An exception indicates that the state is not set correctly.

Comments

This method will determine how diagram control reacts to different user interactions. If DEBUG or ONLINE state is selected, the diagram control will become read only.

5.1.13 IDiagramControl.setDiagramControlProperties

Method

```
setDiagramControlProperties(  
    CDiagramControlProperties diagramControlProps  
);
```

Description

This interface method is called by the containing application to set the properties of the diagram control.

Parameters

Name	Type	Description
diagramControlProps	CDiagramControlProperties	The properties of the diagram control.

Exceptions

Name	Description
DC_FAIL_TO_SET_PROPERTY	An exception indicates that the property is not set correctly.

6 Events

TBD – The diagram control will propagate all the windows events back to the containing view with the same parameter lists if the diagram control catches the events first.

6.1.1 NewFigure

Event

```
NewFigure(CFigureInfo figureInfo)
```

Description

This event will be fired from the diagram control to the containing view when there is a new figure being created.

Parameters

Name	Type	Description
figureInfo	CFigureInfo	Dataset describing the new figure

6.1.2 NewConnection

Event

```
NewConnection(CConnectionInfo connectionInfo)
```

Description

This event will be fired from the diagram control to the containing view when there is a new connection being created.

Parameters

Name	Type	Description
connectionInfo	CConnectionInfo	Dataset describing the new connection

6.1.3 VerifyConnection

Event

```
VerifyConnection(CConnectionInfo connectionInfo)
```

Description

This event will be fired from the diagram control to the containing view when user tries to create a new connection.

Parameters

Name	Type	Description
connectionInfo	CConnectionInfo	Dataset describing the new connection to verify

6.1.4 DeleteFigure

Event

`DeleteFigure(String strFigureName)`

Description

This event will be fired from the diagram control to the containing view when user deletes a figure

Parameters

Name	Type	Description
strFigureName	String	Name of the figure deleted

6.1.5 DeleteConnection

Event

`DeleteConnection(CConnectionInfo connectionInfo)`

Description

This event will be fired from the diagram control to the containing view when user deletes a connection.

Parameters

Name	Type	Description
connectionInfo	CConnectionInfo	Dataset describing the connection deleted

6.1.6 GetFigureContextMenu

Event

`GetFigureContextMenu(CFigureInfo figureInfo)`

Description

This event will be fired from the diagram control to the containing view when user selects on a figure and right clicks to get the context menu.

Parameters

Name	Type	Description
figureInfo	CFigureInfo	Dataset describing the figure whose context menu was requested

6.1.7 GetDiagramContextMenu

Event

`GetDiagramContextMenu(CDiagramInfo diagramInfo)`

Description

This event will be fired from the diagram control to the containing view when user right clicks on the diagram.

Parameters

Name	Type	Description
diagramInfo	CDiagramInfo	Dataset describing the diagram.

6.1.8 NewPosition

Event

`NewPosition(String strName, Point newPosition)`

Description

This event will be fired from the diagram control to the containing view when user moves an object and release the left button.

Parameters

Name	Type	Description
strName	String	Name of the object moved
newPosition	Point	New position of the object

6.1.9 NewSize

Event

`NewSize(String strName, Size newSize)`

Description

This event will be fired from the diagram control to the containing view when user resizes an object.

Parameters

Name	Type	Description
strName	String	Name of the object resized
newSize	Size	New size of the object

7 Datasets

7.1 CFigureInfo

- Dataset describing a figure object

Field Name	Type	Description
Name	String	Figure name (unique within a drawing)
Position	Point	Figure position (coordinates) in the control
Size	Size (readonly)	Figure size; computed based on underlying shapes
FillColor	Color	General figure fill color
ShapeInfos	Collection	Collection of shape infos comprising the figure
ConnectorInfos	Collection	Collection of connector infos
Constraints	CConstraintInfo	Figure constraints

7.2 CShapeInfo

- Abstract dataset describing a shape object

Field Name	Type	Description
Name	String	Shape name (unique within a figure)
NameLayout	LAYOUT_TYPE	Name orientation in the shape
Position	Point	Shape position (coordinates) in the parent figure; relative to the parent figure's position
ZOrder	Int	Shape display order
Label	String	Shape label or caption
LabelLayout	LAYOUT_TYPE	Label orientation in the shape
ShowLabel	Bool	Determines if label should be shown or not
Bitmap	String (URL)	Shape bitmap
RunTimeData	String (readonly)	Shape runtime data
RTDLayout	LAYOUT_TYPE	Runtime data orientation in the shape
ShapeStyle	CShapeStyle	Shape style format

LAYOUT_TYPE – orientation constants

{ LEFT, CENTER, RIGHT, TOP, MIDDLE, BOTTOM, INSIDE, OUTSIDE }

SHAPE_TYPE – shape constants

{ LINE, RECTANGLE, ELLIPSE, CROSS, TERMINATOR }

7.3 CShape_LineInfo

- Dataset describing a line shape; derived from **CShapeInfo**

Field Name	Type	Description
EndPoint	Point	End point (coordinates) of the line figure

7.4 CShape_RectangleInfo

- Dataset describing a rectangle shape; derived from **CShapeInfo**

Field Name	Type	Description
Size	Size	Size (width, height) of the rectangle figure

7.5 CShape_EllipseInfo

- Dataset describing an ellipse shape; derived from **CShapeInfo**

Field Name	Type	Description
Size	Size	Size (width, height) of the ellipse figure

7.6 CShape_CrossInfo

- Dataset describing a cross shape; derived from **CShapeInfo**

Field Name	Type	Description
Size	Size	Size (width, height) of the cross figure

7.7 CShape_TerminatorInfo

- Dataset describing a terminator shape; derived from **CShapeInfo**

Field Name	Type	Description
Size	Size	Size (width, height) of the terminator figure

7.8 CShapeStyle

- Dataset describing a shape style format

Field Name	Type	Description
BorderColor	Color	Outline color
BorderWidth	Int	Width of the border or outline
FillColor	Color	Fill color

7.9 CConnectorInfo

- Dataset describing a connector

Field Name	Type	Description
Name	String	Connector name (unique within a figure)
Position	Point	Connector position (coordinates) in the parent figure; relative to the parent figure's position
Size	Size (readonly)	Connector size; computed based on underlying shapes
FillColor	Color	General connector fill color
ShapeInfos	Collection	Collection of shape infos comprising the connector
Constraints	CConstraintInfo	Connector constraints

7.10 CConnectionInfo

- Dataset describing a connection

Field Name	Type	Description
Name	String	Connection name
StartFigureName	String	Name of the starting figure
StartConnectorName	String	Name of the starting connector
EndFigureName	String	Name of the ending figure
EndConnectorName	String	Name of the ending connector

7.11 CDiagramControlProperty

- Dataset describing the control properties

Field Name	Type	Description
ShowGrid	Bool	Determines if control should show grid or not
SnapToGrid	Bool	Determines if control should snap objects to the grid or not
GridInterval	Int	Spacing between grid points
BackColor	Color	Background color of the control

7.12 CDiagramInfo

- Dataset describing the diagram

Field Name	Type	Description
FigureInfos	Collection	Collection of figure infos
ConnectionInfos	Collection	Collection of connection infos
Constraints	CConstraintInfo	Diagram constraints

7.13 CConstraintInfo

Field Name	Type	Description

Runtime Operator's Workspace

1.	Runtime Operator's Workspace Concept.....	357
1.1	OBJECTIVES	357
1.2	KEY TERMS.....	6
1.3	EXECUTION ENVIRONMENTS.....	7
1.3.1	<i>Single Monitor Workstations</i>	7
1.3.1.1	User needs.....	7
1.3.2	<i>Touch Screens</i>	7
1.3.2.1	User needs.....	7
1.3.3	<i>Keyboard-less Operation</i>	8
1.3.3.1	User needs.....	8
1.3.4	<i>Multi-monitor Workstations</i>	8
1.3.4.1	User needs.....	8
1.3.5	<i>Smart Displays</i>	9
1.3.5.1	User needs.....	9
1.3.6	<i>Tablet PCs</i>	9
1.3.6.1	User needs.....	9
1.3.7	<i>Pocket PC based PDAs</i>	9
1.3.7.1	User needs.....	10
1.3.8	<i>Smart Phones</i>	10
1.4	WORKSPACE DESKTOP MANAGEMENT	10
1.4.1	<i>Dedicated and Controlled Desktop</i>	10
1.4.1.1	User needs.....	11
1.4.2	<i>Another Windows Application</i>	12
1.4.2.1	User needs.....	12
1.5	INTERNATIONALIZATION SUPPORT.....	13
1.5.1.1	User needs.....	13
1.6	WORKSPACE FRAMEWORK (LAYOUT)	14
1.6.1	<i>Framework Panels</i>	14
1.6.1.1	User needs.....	15
1.6.2	<i>Workspace Displays</i>	17
1.6.2.1	User needs.....	17
1.6.3	<i>Workspace applications</i>	19
1.7	USING THE FRAMEWORK	20
1.7.1	<i>Optimized Operation Scenarios</i>	20
1.7.1.1	Respond to Alarm in "Alarm Banner" Panel	21
1.7.1.1.1	Single Monitor Workstation Environment.....	21
1.7.1.1.2	3-Monitor Workstation Environment	26
1.7.1.1.3	Pocket PC Environment.....	29
1.7.1.2	Getting More Information on Display Elements	31
1.7.1.2.1	Single Monitor Workstation Environment.....	31
1.7.1.3	Monitoring Multiple "Hot Issues"	34
1.7.1.3.1	Single Monitor Workstation Environment.....	34
1.7.1.4	Responding to an "Alarm Flood".....	39
1.7.1.4.1	Single Monitor Workstation Environment.....	39
1.7.2	<i>Displays and Workspace applications</i>	42
1.7.2.1	Representing Information in the Same Way	49

1.7.2.1.1	User Needs.....	49
1.7.2.2	Runtime Workspace Health and Connection Status	50
1.7.2.2.1	User Needs.....	50
1.7.2.3	Rearranging Displays and Applications.....	51
1.7.2.3.1	User Needs.....	51
1.7.2.4	Pop-up Dialogs and Menus	53
1.7.2.4.1	User Needs.....	53
1.7.2.5	"Last Display" Features	56
1.7.2.5.1	User Needs.....	56
1.7.2.6	User Preferences and Presentation Persistence.....	57
1.7.2.6.1	User Needs.....	57
1.7.3	<i>Using Displays</i>	58
1.7.3.1	Resolving Display Definitions.....	58
1.7.3.1.1	User Needs.....	58
1.7.3.2	Opening New Displays	61
1.7.3.2.1	Locations for New Displays.....	61
1.7.3.2.2	New Display Scaling.....	61
1.7.3.2.3	Opening Displays from other Displays.....	63
1.7.3.2.4	By Name	63
1.7.3.2.5	From Controls and Workspace Applications.....	65
1.7.3.3	Panning and Controlling Display Scaling.....	65
1.7.3.3.1	User Needs.....	65
1.7.3.4	Interactive Elements in Displays	66
1.7.3.4.1	"Perform Action" Buttons.....	66
1.7.3.4.2	Selection Targets.....	68
1.7.3.4.3	Two State Data Entry Controls.....	70
1.7.3.5	Numeric Data Entry Control.....	71
1.7.3.6	Alphanumeric Data Entry Control.....	71
1.7.3.7	Specialized Display Components	72
1.7.3.7.1	Pop Up Dialog Displays.....	72
1.7.3.7.2	Sub-Panel Displays	72
1.7.3.7.3	Operator Prompt.....	72
1.7.3.7.4	Alarm Flood Management.....	72
1.7.3.7.5	Display Tree.....	72
1.7.4	<i>Using Workspace applications</i>	72
1.7.4.1	System Node Status	72
1.7.4.2	DeltaV System Diagnostics	72
1.7.4.3	History Views	72
1.7.4.4	Batch Operations.....	72
1.7.4.5	Module Operation Details (On-line).....	72
1.7.4.6	DeltaV Inspect	72
1.7.4.7	(AMS) FF & HART Device Details.....	72
1.7.4.8	Asset Alert Details	72
1.7.4.9	Advanced Control Operations.....	72
1.7.4.10	DeltaV Safe Operations	72
1.7.4.11	Media Gallery	72
1.7.4.12	Display Alerts	72
1.7.4.13	Runtime Workspace Diagnostics.....	72
1.7.4.14	Document Viewers.....	72
1.7.4.15	Web Browser	72

1.8	SECURITY FEATURES	72
1.8.1	<i>User Authentication</i>	72
1.8.2	<i>Fast User Switching</i>	72
1.8.3	<i>User Privilege Aggregation</i>	72
1.8.4	<i>Electronic Signatures</i>	72
1.8.5	<i>Inter-zone Credentials Wallet</i>	72
1.9	PRE-ENGINEERED FRAMEWORKS	72
1.9.1	<i>Single Monitor Framework</i>	72
1.9.1.1	Tool Bar Display	72
1.9.1.2	"Main Display" Model.....	72
1.9.2	<i>Dual Monitor Framework</i>	72
1.9.2.1	Tool Bar Display	72
1.9.2.2	"Main Display" Model.....	72
1.9.3	<i>Pocket PC Framework</i>	72
1.9.3.1	Tool Bar Display	72
1.9.3.2	"Main Display" Model.....	72
1.9.4	<i>Smart Phone Framework</i>	72
1.9.4.1	Tool Bar Display	72
1.9.4.2	"Main Display" Model.....	72
1.10	PRE-ENGINEERED DISPLAYS	72
1.10.1	<i>Alarm Banners</i>	72
1.10.1.1	Single Monitor Alarm Banner	72
1.10.1.2	Dual Monitor Alarm Banner(s).....	72
1.10.1.3	Pocket PC Alarm Banner	72
1.10.2	<i>Alarm Summary Displays</i>	72
1.10.2.1	All Alarms Summary	72
1.10.2.2	All Suppressed Alarms Summary	72
1.10.2.3	Area Filter Display.....	72
1.10.3	<i>Faceplates</i>	72
1.10.3.1	Module Faceplates	72
1.10.3.2	FF Device Faceplate	72
1.10.3.3	Logic Solver Faceplate	72
1.10.4	<i>Detail Displays</i>	72
1.10.5	<i>DeltaV System Node Status Display</i>	72
1.11	RUNTIME WORKSPACE DIAGNOSTIC FEATURES	72
1.11.1	<i>Diagnostic Operation Levels</i>	72
1.11.1.1	Minimal.....	72
1.11.1.2	Normal	72
1.11.1.3	Heightened	72
1.11.1.4	"In Your Face" (Message Dialog Notification)	72
1.11.2	<i>Recent Incident/Information Logs</i>	72
1.11.2.1	Parameter Read Failures	72
1.11.2.2	Parameter Write Failures	72
1.11.2.3	Display Logic Trace Output.....	72
1.11.3	<i>Display Logic Diagnostic Execution Log</i>	72

1. Runtime Operator's Workspace Concept

As a part of the development program, this document describes the concept for collection of applications that provide the primary operator interface for the DeltaV system.

It's goal is to identify user needs in this area, and to define a concept for the set of system features that would meet those needs.

This concept document defines the primary objects and the key actions that can be performed on those objects, which users will need to understand in order to utilize the runtime workspace..

1.1 Objectives

The objectives for the runtime workspace and applications are to provide superior primary operator interface tools for the DeltaV system:

1. Highly reliable, "mission critical" application intended for process control operators who need to stay in control of the "control system" in all circumstances.
Characteristics include:
 - Always available. In many cases, the application may be available for long periods of time (e.g. weeks, months, even years) without unplanned disruptions.
 - Always interacting. Operator may always have the opportunity to "move on"; getting predictable responses to new directives even when previous requests take "too long" to complete, or "everything is changing" due to a plant upset.
 - Mistake resistant. Repeatable behaviors and user interaction features that can help reduce the possibility of high cost "human errors"; especially in high stress situations.
2. Extremely responsive interactions and high performance display of process information. Expectations for large user configured displays to appear and be populated with live data from the DeltaV system typically within 1 second. (Smaller displays like our pre-engineered "faceplate" style displays correspondingly faster.)
3. Supports a dedicated (kiosk style) operator environment, which is extremely resistant to users damaging the programs or data present on that workstation, or gaining access to unintended applications. The degree of restriction is user configurable, so that the workspace is also suitable for use by trusted and higher skill level users.
4. Adaptable to a wide range of user interface hardware features:
 - 1- 2- 3- or 4- monitor video configurations
 - 4:3(PC format) and 16:9(HDTV format) aspect ratio monitors
 - mouse + keyboard operation
 - touch screen operation
 - PDA form factors and user input techniques
 - Tablet PC form factors and user input techniques
 - Smart Phone form factors and user input techniques

5. Excellent integration of operator tools/applications (e.g. Batch, diagnostics, history viewing) with user configurable displays (and user display logic). Easy access to information related to display elements. Easy access to auxiliary information sources (e.g. video feeds) and collaboration technologies (e.g. e-mail and instant messaging.)
6. Operator-friendly tools to select and manage the content in the runtime workspace; minimizing window management responsibilities. Improved features for rendering content within the available space, so that a content sources not specifically engineered for the workspace are easily usable.
7. Well integrated user security and authentication facilities to support:
 - Fast DeltaV user switching (comparable to previous systems)
 - DeltaV user collaboration (temporarily aggregating security privileges)
 - Multi-user authentication for electronic signature requirements
 - Authentication technologies that do not require passwords
 - Login timeouts
8. Platform for improved information capture and analysis tools for operations staff. Features like:
 - Snapshot log: recording information for later examination
 - Tools for finding revealing alarms in an alarm flood
 - "Back in time" displays: ability to look at displays based on (automatically collected, recent) historical data

1.2 Key Terms

Term	Description
Display definition	Data (file) that holds all the configurable properties and the definition and layout of all the elements used to create an instance of a workspace display . Most display definitions are created by the display configuration application(s).
Framework panel	Rectangular display areas within the workspace framework , where workspace displays or workspace applications can appear.
Runtime workspace	The interactive DeltaV application(s) that provide the operator interface environment.
Workspace application	"Built in" DeltaV applications designed to work within the workspace framework , showing information and supporting user interactions through framework panel(s) .
Workspace display	Highly user-configurable content in framework panels that typically shows updating process information (DeltaV parameters) and graphic schematics of process equipment under automated control. Also capable of interactions (changing parameter values, opening new displays, starting workspace applications , etc.) An instance of an active workspace display is created from a display definition .
Workspace framework	The configurable layout of the runtime workspace ; how it will appear on display hardware. Composed of framework panels .

1.3 Execution Environments

We expect the runtime workspace to be well suited for a number of computing platforms and execution environments.

1.3.1 Single Monitor Workstations

For the foreseeable future, the runtime workspace will most often be deployed on single monitor PCs. This will continue to be a cost-effective (and best performing) platform for primary and casual "operator" interfaces for the DeltaV system; often the best choice where flexible mounting or limited space issues are present.

1.3.1.1 User needs

- a) A single display device, approx 4:3 aspect ratio (minimum pixel resolution of approx 1024x768), or 16:9 aspect ratio (minimum pixel resolution of approx 1280 x 720).
- b) Standard PC keyboard and two button pointing device (usually a mouse) for all interaction.
- c) All major features of the DeltaV primary operator interface may be available in the single monitor configuration¹.

1.3.2 Touch Screens

We can expect a significant number of users wishing to use the runtime workspace with touch screen hardware; especially when conditions are not suitable for a mouse as a pointing device. Touch screens typically make certain interactions more difficult than they are with a mouse:

- Hitting closely packed small buttons or list items accurately (thus more mistakes are quite likely)
- Features depending on mouse pointer position feedback (e.g. tool tips, context menu selection feedback, etc.)
- Operations requiring "dragging"
- 2nd button (right click) and 3rd button/wheel operations

1.3.2.1 User needs

- a) No operations that require a 3rd button or mouse wheel
- b) User configurable "touch screen" preferences that can be applied per workstation/session:
 - Confirmation step required before any "single click" parameter writes
 - "Point to move; double point to click" preference for pointer behavior
 - Disable drag operations (implies alternatives are available)
 - Right button alternative (e.g. press and hold time)
 - Large vs. small buttons in workspace applications
 - Font size for selection lists
 - Large (button layout) vs. small (space efficient) preference for workstation dialogs
- c) User configurable large and small button choices in display controls; adequate button spacers.
- d) Recommended (and tested) windows appearance settings for touch screen use

¹ After all, unless a demo room is handy, the single monitor (laptop) is probably the platform where customers will be first exposed to the runtime workspace.

1.3.3 Keyboard-less Operation

We should expect users to sometimes deploy the runtime workspace on hardware tailored for harsh/dirty industrial environments; situations unsuitable for a standard keyboard, and a hardened keyboard is not cost effective. In other applications, operator interaction requirements may be quite limited or infrequent, and the need for a keyboard is not justified considering the mounting conditions.

1.3.3.1 User needs

- a) Virtual (on screen) keyboards or numeric keypads can be utilized to perform all interactions via a pointing device (including touch screen); eliminating the need for a hardware keyboard.
- b) User configurable "virtual keyboard" preferences that can be applied per workstation/session:
 - Automatically display numeric keypad with all numeric data entry controls (with large or small buttons)
 - Automatically display virtual keyboard with all alphanumeric data entry controls (with large or small buttons)

1.3.4 Multi-monitor Workstations

Process operation problems often cannot be immediately dispatched. They often require operators to go through an assessment/make changes/monitor results cycle, that can proceed at the pace of process dynamics, and sometimes may need fine adjustments along the way.

When an operator has several problems occurring at the same time, (s)he is faced with staying on top of multiple conditions, progressing at different rates, and probably not in the same plant area/unit (and thus not combined in any single display.) Having a larger "workspace", capable of simultaneous display information from several "problem areas" can significantly help operators with their responsibilities.

Anticipating the low cost and significant operational benefits of multi-monitor workstations we should expect increasing use of multi-monitor hardware as the primary "control room" operator's workstation for DeltaV systems.

1.3.4.1 User needs

- a) The ability to utilize multi-monitor capabilities to show more information, and give operators more control over the combination of information on display.
- b) A user configurable means to partition the available monitor screen area (a "framework") so that:
 - Displays designed for single monitor workstations can be used effectively on multi-monitor workstations.
 - New displays that are called up automatically go to the "default" place, and "fit"; without the operator having to manage windows or direct display placement.
- c) Allowing operators to easily control and arrange the combination of displays that appear; helping him monitor information related to his current concerns.

1.3.5 Smart Displays

With Windows powered Smart Display devices, wireless communication (rather than cables) connect the human interface hardware (monitor/touch screen/keyboard/pointing device) to the "operator workstation" PC. DeltaV users may find this technology valuable for:

- Operations supervisors and other plant management staff with needs to interact with the DeltaV system from offices and conference rooms.
- Roaming "helper operators" in large control rooms.
- Better able to restrict physical access to critical server workstations (e.g. the PRO+, terminal servers workstations, etc.)

Smart Displays generally have touch sensitive screens, on screen keyboards, handwriting recognition and sometimes (optional) keyboard and mouse connections.

1.3.5.1 User needs

(It would seem that the support for Smart Displays introduces no additional user needs, not already required by touch screen and keyboard-less operation discussed above.)

1.3.6 Tablet PCs

Tablet PCs (like laptop PCs) integrate the "human interface" hardware (monitor) with the (Windows XP capable) "workstation hardware", but add touch sensitive screens, stylus based interactions, handwriting recognition, and on-screen keyboards. Users may find this technology valuable for:

- Occasional use, remote "control rooms". (Any closet or shack with a work surface, power, and a network connection (or wireless access point near by) can become a temporary control room.)
- Very portable DeltaV Simulate systems.

Users would probably choose Tablet PCs over Smart Displays/server considering:

- a) Snappier display interaction performance (no wireless comms between application and display)
- b) Integrated HW is probably less expensive.
- c) They really want a portable personal computer first; and DeltaV runtime workspace platform as a secondary issue.

1.3.6.1 User needs

- a) Given the potential for intermittent wireless connections, a means to let the user quickly know if the data on display is "fresh" or is "stale" due to communications breaks.
- b) Support for Tablet PC-style handwriting recognition as a means to complete interactions that involve data entry.

(In other respects, it would seem that the support for Tablet PCs (with the runtime workspace software installed on them) would have similar user needs as those already required by single monitor, touch screen and keyboard-less operation discussed above.)

1.3.7 Pocket PC based PDAs

The size and cost factor for Pocket PC PDAs gives them the potential to be a popular platform for interacting with DeltaV systems. Integrated microphone/speaker capabilities offer potential for entry and playback of audio commentary. PDA usage possibilities include:

- System/device health status monitoring by (highly mobile) supervisory or maintenance staff.
- Real-time access to business (production volume, yields, inventory levels) and/or environmental information available from the DeltaV system.
- Ubiquitous access to reference documents (safety procedures, shutdown checklists, material safety data sheets, plant drawings, etc.)
- Task specific operator interfaces for roaming operators (verifying valve positions and sensor readings, "watching" a unit as maintenance is performed nearby, etc.)
- Field device identification (activating the "flash your LEDs" mode to verify physical location) and repair operations.
- Entering comments (including audio) on process equipment during inspection rounds.

PDA device screen size (<10% (pixel-wise) of a single "full size" monitor) places significant constraints on how they can be effectively used with DeltaV systems. Memory limitations along with the administrative effort to install/maintain correct (even multiple?) versions of DeltaV-specific software on hand held devices, suggests minimizing the amount of special software that may be maintained on the PDAs².

1.3.7.1 User needs

- a) Configurable workspace frameworks and displays suitable for PDA screens.
- b) Alarm banner and alarm summary controls designed for very small screens and stylus style interactions
- c) Operator comment (data and voice) entry applications³. (Probably optimized so that "stock" comments or log entries can be entered with minimum virtual keyboard use.)
- d) Minimum software installation and version management administration for PDA users.

(In other respects, it would seem that the support for PDA devices would have similar user needs as those already required by single monitor, touch screen and keyboard-less operation discussed above.)

1.3.8 Smart Phones

1.4 Workspace Desktop Management

The runtime workspace application(s) will operate in either of the following desktop management modes:

- The "Dedicated and Controlled" desktop
- As just "Another Windows Application"

1.4.1 Dedicated and Controlled Desktop

The "dedicated and controlled" desktop management mode allows DeltaV system administrators to deploy the runtime workspace so that its users do not inadvertently (or maliciously) damage the software/data installed on that (serving) workstation, nor cause the runtime workspace to be inoperative.

² At this point, use of terminal services technology, thus requiring a (pretty stable) terminal service client application on the PDAs, seems like the thinnest possible "thin client".

³ Implies operator comment storage infrastructure.

1.4.1.1 User needs

- a) It is possible to configure a workstation (or session on a server) to automatically start after system boot-up in "dedicated and controlled" desktop management mode. In this mode, one instance of the runtime workspace is allowed to run in each workstation (session).
- b) A "workspace hard reset" mechanism is required, which causes the run-time workspace to revert to the initial startup condition (framework contents) without requiring it to be shut down and manually restarted. This is intended as the last resort available to operators to restore an (apparently) malfunctioning runtime workspace to working condition (short of rebooting the workstation/server). It is acceptable for a "workspace hard reset" to take up to 30 seconds⁴.

During the reset, a message should be provided indicating that a reset is in progress, along with some updating indication that the screen hasn't frozen.

- c) Certain users should not be constrained to the "dedicated and controlled" desktop management mode. Users with appropriate DeltaV security keys, can switch to "another Windows application" desktop management mode (retaining as much current workspace context (panel contents, recently used history, etc.) as possible.
- d) While in "dedicated and controlled" desktop management mode, users are prevented from running unintended programs or applications. This requires:
 - Disabling access to the Start dialog (Windows key and Ctrl-Esc)
 - Disabling access to the Windows taskbar
 - Disabling access to Windows keyboard shortcuts; especially including:
 - a. Run dialog (WinKey + R)
 - b. Minimize all (WinKey + M)
 - c. Switch to another (non-runtime workspace) application (Alt-tab)
 - d. OS Explorer (WinKey + E)
 - Not allowing access to Windows desktop shortcuts
- e) The runtime workspace will allow specific (non-runtime workspace) applications that have been authorized (via DeltaV configuration). The user will be presented a list of applications (and descriptions) to choose from⁵.
- f) While in "dedicated and controlled" desktop management mode, users are prevented from making the runtime workspace inoperative. This requires:
 - Not allowing the termination of the (primary) runtime workspace application (e.g. Alt-F4, or Exit menu items)
 - Disabling access to the Windows Security dialog (Ctrl-Alt-Esc)
 - Disabling access to Windows keyboard shortcuts; especially including:
 - a. Minimize all (WinKey + M)
 - b. Lock workstation (WinKey + L)

⁴ The intent that this would be a "deep" enough reset to completely clean up and reinitialize the runtime workspace and all runtime workspace companion applications.

⁵ Configurers are responsible for choosing non-runtime workspace applications to run that don't allow the operator to alter or delete files, launch still other "unconstrained" applications, etc.

- Disabling access the Windows Display Properties dialog (e.g. access to altering display color depth and resolution settings, appearance preferences, themes, wallpaper, etc.)
- g) The runtime workspace will provide a "constrained Internet browser". While in "dedicated and controlled" desktop management mode, web pages from authorized (via DeltaV configuration) URLs will be displayed or accessible via hyperlinks⁶.
 - h) While in "dedicated and controlled" desktop management mode, any screen savers are disabled.
 - i) Runtime workspace compliant applications do not provide access to the Windows folder and file properties dialog when performing any file browsing operations. They do not allow files to have their properties or security requirements changed, or be renamed or deleted, unless doing so cannot possibly damage the software or data installed on that workstation.

1.4.2 Another Windows Application

The "another Windows application" desktop management mode allows the runtime workspace to be used in conjunction with other Windows applications, by appropriately authorized and skilled personnel. Switching to this mode is useful for:

- using the DeltaV system configuration applications (including workspace and display configuration applications)
- gaining access to debugging features not appropriate for all users
- troubleshooting the runtime workspace

1.4.2.1 User needs

- a) To aid runtime workspace troubleshooting, the "another Windows application" desktop management mode supports all the features and interactive behaviors of the "dedicated and controlled" desktop management mode as much as possible. Conceptually, it just removes constraints and adds extra features.
- b) While in the "another Windows application" desktop management mode, the user has access to the normal Windows desktop application management features; including:
 - Task bar, Start button, Run dialog, etc.
 - Windows key and Windows key shortcuts
 - Application minimization (including the runtime workspace)
 - Application switching (Alt-tab)
 - Unrestricted ability to change display properties
 - The ability to terminate the runtime workspace application(s)
- c) While in the "another Windows application" desktop management mode, there is a mechanism to "spin off" a new (detached) Windows application window on the Windows desktop, with the same content (display or runtime workspace application) as any of the framework panels (fixed or floating). This gives the user the ability to easily create content in new windows, where s(he) is responsible for window management (positioning, resizing, minimizing/maximizing, scrolling content within

⁶ Goal here is to support users who maintain documents and web pages on intra-network servers; and still want to prevent uncontrolled roaming on the Internet.

the window client area, closing, etc.) Once such windows are created, they no longer interact with or "track" actions in the runtime workspace framework.

- d) With access to the Windows desktop, users may "run" new instances of the runtime workspace application; choosing for it to start up in either "dedicated and controlled" or "another Windows application" desktop management mode:
 - When launched in "dedicated and controlled" desktop management mode, the Windows desktop will be controlled by the runtime workspace application, and other running applications will (probably) be inaccessible⁷.
 - When launched in "another Windows application" desktop management mode, the user may run multiple instances of the runtime workspace application⁸.
- e) All users may switch from the "another Windows application" desktop management mode to the "dedicated and controlled" desktop management mode (no security key is required.) The switch retains as much current workspace context (panel contents, recently used history, etc.) as possible.

1.5 Internationalization Support

Today, most "operations" features of the DeltaV system are available, localized to one of several languages. The runtime workspace, being the primary "operations" application, may continue to support the current language choices. It should also be expected that not all workstation/terminals/PDAs connected to a DeltaV system will want to have the runtime workspace use the same language choice⁹.

Also, should technical support from EPM/AUS be needed to consult on a non-English DeltaV system, the English speaking technical support staff has the option of switching to, or running a separate English runtime workspace to facilitate communications¹⁰.

1.5.1.1 User needs

- a) Users may choose one or more of the available DeltaV language options when installing DeltaV on runtime workspace-capable workstations or terminal servers. One language is identified as the "dominant" language for the system; determining the standard and default configuration values for that DeltaV system. The dominant language may be the same for all DeltaV workstations in the same system.
- b) When an instance of the runtime workspace is started, the dominant language is automatically chosen by default, and all standard workspace behaviors and interactions use that language.

⁷ The user should probably close (or park) them appropriately before starting up a "dedicated and controlled" runtime workspace.

⁸ For example, taking advantage of multi-monitor hardware to use/test several single monitor display frameworks running in different instances of the runtime workspace application. Or perhaps run an English instance of the runtime workspace in parallel with a non-English instance, to assist English speaking troubleshooters.

⁹ Since most engineering and administrative applications are available in English, it seems quite likely that some users will want to run English runtime workspaces in non-English DeltaV systems. (Seems like this could facilitate technical support as well.)

¹⁰ We may want to consider always installing English secondary language support on workstations in non-English DeltaV systems.

- c) It is possible to override the language choice for the runtime workspace with additional information provided at start up. It is also possible for a user to switch to another language for an existing runtime workspace instance by selecting one (of those installed on this workstation) in the course of doing a "workspace hard reset" operation.
- d) When the workspace is running using a language other than the dominant language for the system, the workspace applications provide an interface (menus, dialogs, etc.) localized for the chosen language. Information content that is determined by the DeltaV system configuration, such as:
- Area, module, node, DST, FF device names
 - Module, node, FF device descriptions
 - User enum state names
 - Alarm parameter names, alarm words, priority words
 - Batch procedure/recipe names, grade names
- ...still retain their configured names (presumably compatible with dominant language.)

Also, information stored in common system databases (available to multiple workstations/sessions) such as:

- Event Chronicle databases
- Batch Historian databases
- Continuous History databases
- Device Audit Trails

...remain, and are displayed using the dominant language for the system.

1.6 Workspace Framework (Layout)

The workspace framework provides a user configurable layout for how information is arranged within the runtime workspace; utilizing the available display area on the target display hardware. We anticipate:

- Users will often choose one of the pre-engineered frameworks that are part of the default DeltaV configuration. Users (or system integrators) with special requirements or unique display hardware will design/configure their own framework(s).
- Once a framework is chosen, it will be widely used within a DeltaV system (across many workstations/terminals having similar display hardware). This helps makes operators be more effective wherever they are working; and helps reduce training time and mistakes.
- Displays will be designed with the framework (panel) dimensions in mind; building displays that will "nicely fit" the panels without a lot of "stretching" distortion or having display elements end up being too big (wasting precious space) or too small (to read easily).
- Even when different display hardware configurations are used in the same system, users will strive to use similarly sized panels across their frameworks, so that the same displays will render nicely on each hardware configuration; avoiding the effort of engineering separate displays for each.

1.6.1 Framework Panels

Framework panels are rectangular display areas where runtime workspace displays or applications can be made to appear. There are two types of framework panels:

- **Fixed panels:** have a fixed location in the framework and area always visible (even if they have no content). Are not allowed to overlap other fixed panels. The collection of fixed panels in the workspace framework is the visible "background surface" of an instance of the runtime workspace application.
- **Floating panels:** provide a means for a controlled number of temporary content windows that float over (obscuring) other panels. When the content in a floating panel is closed, the floating panel disappears. Floating panels may be configured to be movable: allowing the operator to move the floating panel away from it's anchor point, giving him more control¹¹ over the visible portions of all panels.

1.6.1.1 User needs

- a) Framework panels are capable of showing several types of content at any point in time:
 - User configurable runtime workspace (graphic) displays.
 - Runtime workspace applications.
- b) Floating panels in the workspace framework have a configurable "use order". When two closed floating panels are otherwise equally eligible to be selected to display new content, the floating panel with higher precedence in the use order will be chosen.

Fixed panels have a separate configurable "use order" to establish precedence.

- c) Through configuration, it is possible to categorize workspace display definitions and framework panels so that displays of a certain category will be automatically directed to appear in a fixed or floating panel with a matching category name¹² when a destination panel is not otherwise specified.
 - One framework panel (fixed or floating) may be configured with 0 or more category names.
 - A display definition may be configured with 0 or 1 category names.
- d) It is possible to assign the same category name to several (fixed or floating) panels. If a new display's category matches several several panels, the display is directed per the sequence:
 - If a fixed panel that currently has no content matches the new display's category, the the new display will replace the old content in that fixed panel. Fixed panel use order will resolve which fixed panel to use if more than one is eligible.
 - If a floating panel that is currently closed matches the new display's category, that floating panel will be opened with the new display content. Floating panel use order will resolve which fixed panel to use if more than one is eligible.
 - If a floating panel that is currently open matches the new display's category, that floating panel's content will be replace with the new display. If more than one open floating panel matches, the one with the oldest content will have it's contents replaced with the new display¹³.
 - If a fixed panel matches the new display's category, the new display will replace the old content in that fixed panel. If more than fixed panel matches, the one with

¹¹ At the cost of more window management responsibility.

¹² Users will tend to define display category names that reflect a combination of native display size and purpose.

¹³ Supporting "round-robin" distribution of floating displays (of certain categories) to their corresponding floating panels.

the oldest content will have its contents replaced with the new display.

- e) It is possible to configure panels to be the automatic "destination" for the workspace applications (e.g. diagnostics, batch operations, history view, etc.)
- f) One panel (either fixed or floating) in each framework is the "default destination" panel. When no specific panel is identified for new framework content and there is no category name match between the new content and any panel, new content appears on this default destination panel.
- g) When an instance of the runtime workspace is first started (or goes through a "workspace hard reset") initial fixed panel contents are displayed. Each fixed panel may have an "initial content" (e.g. workspace display, or application) defined in the framework configuration. The configured initial content for fixed panel(s) may be overridden with information provided at runtime workspace startup time.
- h) Framework fixed panels may be configured to have several border options to help delineate them from other fixed panels in the framework. Options include "no border".
- i) A floating panel's upper left corner initially appears at a configurable "anchor point" position in the workspace framework.
- j) Each floating panel is configured with a preferred size (height and width). It is also configured with a preference for how to choose the initial size:
 - Use native size: the native size of the display establishes the initial size of the floating panel window. (When the content has no "native size" (e.g. workspace applications) the floating panel's preferred size is used as the initial size. The floating panel display window will be constrained to not go beyond the workspace framework.
 - Use preferred size: the panel content is scaled (within any scaling constraints defined for the workspace) to fit within the preferred floating panel size, with possible gaps or scroll bars.
- k) Each floating panel is configured to be resizable or not. If the floating panel is configured to be resizable, its window is rendered with a border suitable for use in window resizing operations. If the floating panel is not configured to be resizable, window resizing operations are not supported (the floating panel will always remain at its initial size.)
- l) Some panels in a framework require a mechanism for user interactions with the panel. These include:
 - Closing the panel contents (e.g. floating panels)
 - Moving a floating panel
 - Picking the panel as the source or destination for a "copy contents to another panel" operation.
 - Panel content scaling or zoom operations.
 - Recall recent panel contents operations.

- Open contents as new (Windows task bar) window¹⁴.
- Capturing a "snapshot" of the current contents.
- Controlling the display of display layers.
- Controls involving the "display using historical data" feature
- Resizing floating panels (without dragging borders)
- (other operations introduced in the future)

Also, depending on available space, users may want title or caption information about panel contents, for example:

- Display name
- Currently selected object (e.g. module, node) name

Framework panels may be configured to have "information and tool button" bars. The content of these bars is configurable. When display area is extremely limited, a mechanism is available to let the info/tool bar to be hidden or minimized.

1.6.2 Workspace Displays

Workspace displays are the mechanism to provide very customized content (and supporting display-related logic) within the runtime workspace. Workspace displays are intended to be used for:

- System specific "process graphic" displays; showing live process information and providing a means to manipulate process parameters.
- Logical "control panels" (or "faceplates") for DeltaV modules, nodes, etc.
- User customizable "pop-up dialogs"
- System status and alarm summary information
- Plant drawings or other documentation that may be annotated with live process information.

*(For details on what runtime workspace displays can be composed of, see the "**Process Display Primitives and Components**" concept document.)*

*(For details on the features and capabilities of display-related logic, see the "**Process Display Logic and Languages**" concept document.)*

1.6.2.1 User needs

- a) Workspace displays have names used for identifying them:
 - Users selecting them from display list/tree browsers, wanting to find/open a display by name.
 - Used by display elements (like display launch buttons) to open another display
 - Used by display logic to pick the appropriate display for the current conditions
- Restrictions on display names should allow:
- Them to be long enough to serve as their own description in a list of display names (allowing for localized names)
 - Characters that can server as delimiters (e.g. space, hyphen, underscore, etc.)
 - Upper/lower case preservation, but case-insensitive comparisons.
 - Easily reversible conversions to/from legitimate file system names

¹⁴ When the workspace is in "another windows application" desktop management mode.

- b) We can anticipate users defining a large number of displays for their DeltaV system. To help provide organization of these for the operator, a hierarchical folder/directory system should be supported. Since folder/directory names in such a system will serve as their own description, the restrictions on these names should be comparable to those on display names.
- c) Some workspace displays (especially those created by importing process instrumentation drawings from non-DeltaV applications) are likely to contain display elements¹⁶ that are not always appropriate for display under typical conditions. Yet, for special conditions (or when used by engineering or maintenance personnel) getting access to this information would be valuable.

Workspace displays will support display elements associated with "display layers"¹⁷. The display configurator can segregate information into separate layers, and choose which layers are visible by default when workspace displays are first rendered in the runtime workspace.

The runtime workspace will provide a mechanism, at the framework panel level, to indicate what layers are available in a display, and change which layers currently appear. For each layer, three levels of visibility is supported:

- Hidden: display elements associated with that layer are not rendered.
- Subdued: display elements associated with that layer are "ghosted" (configured level of transparency) behind any full view layers.
- Full view: display elements are rendered with their normal (configured) color and transparency level (usually opaque)

- d) Workspace displays may be configured to offer special display layers automatically created by the runtime workspace. Special display layers provide an mechanism to add "smart features" to user configured displays with little extra display configuration work. Special display layers that should be considered include:
 - Auto "module controls": tiny module "faceplates" appear on top of display elements with data links to module parameters. (Automatically "weeding down" to at most one faceplate per module.)
 - Auto "trend views": small, semi-transparent real-time trend windows appear next to display elements involving numeric data links. Trend windows support interactions to quickly delete unwanted ones, drag to better position, increase size, make opaque, or launch the a full function process history view application in the appropriate panel.
 - Auto "alarm volume history": for each plant area and Unit associated with data links on the display, an active alarms by priority vs. time column chart is displayed; indicating when and where alarm bursts occurred.
- e) While the "fixed and controlled" nature of the runtime workspace framework is intended to relieve operators of most window management activity, for some users it may be too rigid. Resolving and monitoring different process problems may call for

¹⁶ Information appropriate to the original drawing, but has little to do with the DeltaV system and the automatic process controls; e.g. piping tags, size and materials, manually operated valves, visual inspection points, etc.

¹⁷ Similar in concept to the "drawing layers" features in CAD applications.

more flexibility in using the available display space.

Specialized "sub-panel" workspace displays may be configured. Though capable of containing typical display elements, they are distinct in that they contain panels that support the configuration and behaviors of framework fixed panels.

Sub-panel displays offer a couple of features to help users better tailor the runtime workspace to the problems of the moment:

- By choosing a sub-panel display, the operator can dynamically subdivide one of the panels in the framework into a pre-defined arrangement of smaller fixed panels. (Consider sub-panel display intended to subdivide the "main" display panel, into a row of "faceplate" sized panels on top, and a wide panel ideal for a trend window on the bottom.)
- Once the panel contents have been defined in a sub-panel display, the operator can save the display (in temporary user-session display storage) for later use in the session. Essentially, instances of new displays, dynamically composed to address specific process monitor problems, can be created and reused by that user/workstation/session.

1.6.3 Workspace applications

Workspace applications are the means to deliver task-specific, optimized, "ready out-of-the-box" user interfaces via the runtime workspace. These user interfaces require little or no engineering or customization between DeltaV systems. They also offer an opportunity for richer user interfaces; ones that would be difficult to accomplish through combining display primitives and components and display logic.

Workspace applications differ from normal Windows applications in that:

- They communicate with and their lifetime is controlled by the runtime workspace.
- Expecting not to be "full screen" applications, they take particular care to automatically adjust their user interface layout to best fit the panel area they are assigned to work within. When used in a floating panel, they make every attempt to provide a suitable layout within the initial preferred size configured for the floating panel. As the floating panel is resized, they adjust their layout accordingly.
- They follow the runtime workspace user interface conventions for common actions (e.g. selecting, scrolling, scaling, etc.) so the entire workspace feels well integrated.
- They follow workspace conventions for retaining user interface state, so that "go back to last display" operations for panels work well.
- Where possible they offer useful "tracking" modes of operation; where their content/focus automatically follows the system object (module, node, FF device, batch, logic solver, etc.) that has focus in other panels in the framework (usually the "main" or "default destination" panel). They offer a means to disable tracking mode.

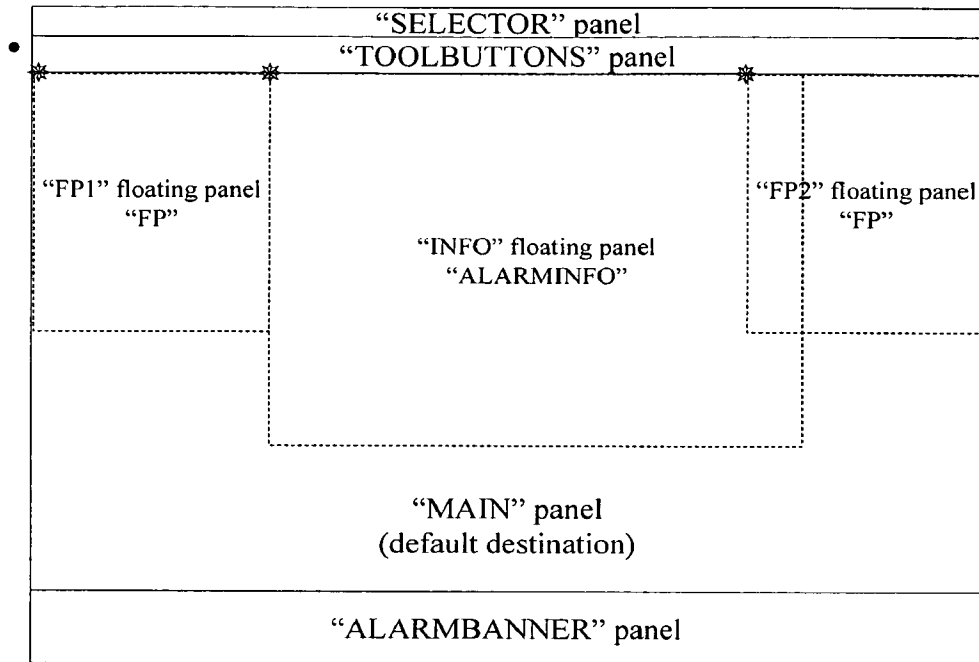
We would expect that adding new workspace applications would be used frequently as the capabilities of the DeltaV system increases. (See details on each using each workspace application in later section.)

1.7 Using the Framework

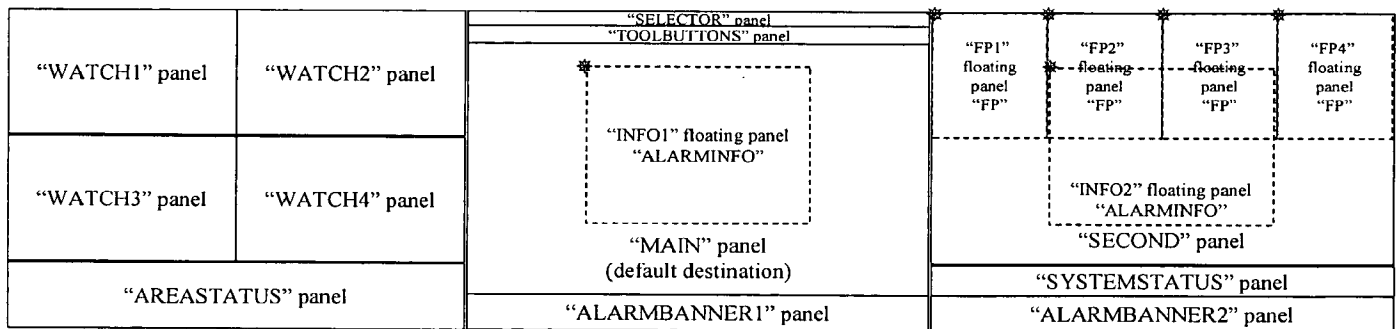
1.7.1 Optimized Operation Scenarios

This section explores runtime workspace features aimed at providing excellent facilities for handling frequent operator tasks.

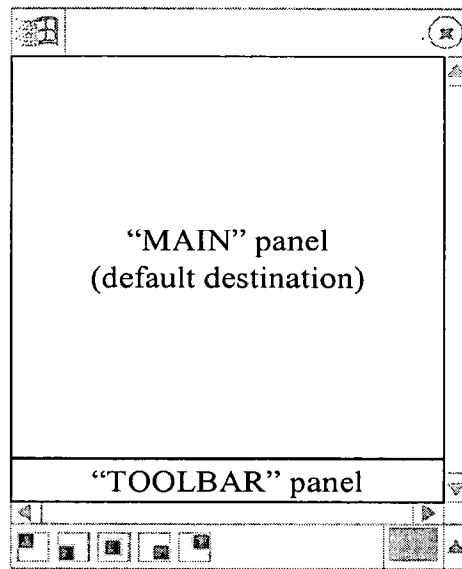
The scenarios assume the following runtime workspace environments and frameworks:



- The a 3-monitor workstation environment and workspace framework:



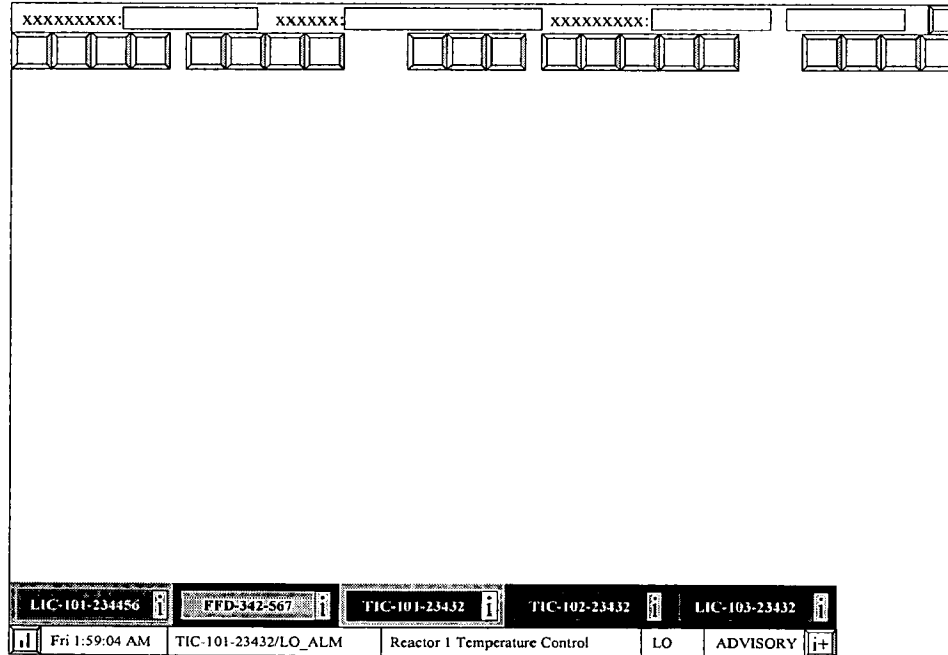
- A Pocket PC terminal session environment and workspace framework:



1.7.1.1 Respond to Alarm in "Alarm Banner" Panel

1.7.1.1.1 Single Monitor Workstation Environment

1. System has been quiet. Operator has several alarms appearing in the alarm banner, but has checked them out and is confident that they will be resolved later in his shift.



2. A moderate priority alarm occurs in Reactor 1. The operator is alerted to it first by the workstation making the "WARNING" level alarm (horn) sound. He sees the new alarm in the alarm banner. The alarm banner buttons have backgrounds indicating age of the alarm (white for very recent, to black for alarms that have been active for > 8 hours). He quickly know that the one white background alarm has occurred very

FIC-101	LIC-101-234456	FFD-342-567	TIC-101-23432	TIC-102-23432
Fri 10:23:34 AM	FIC-101/HI_ALM	Reactor 1 Primary Inlet Flow	HI	WARNING

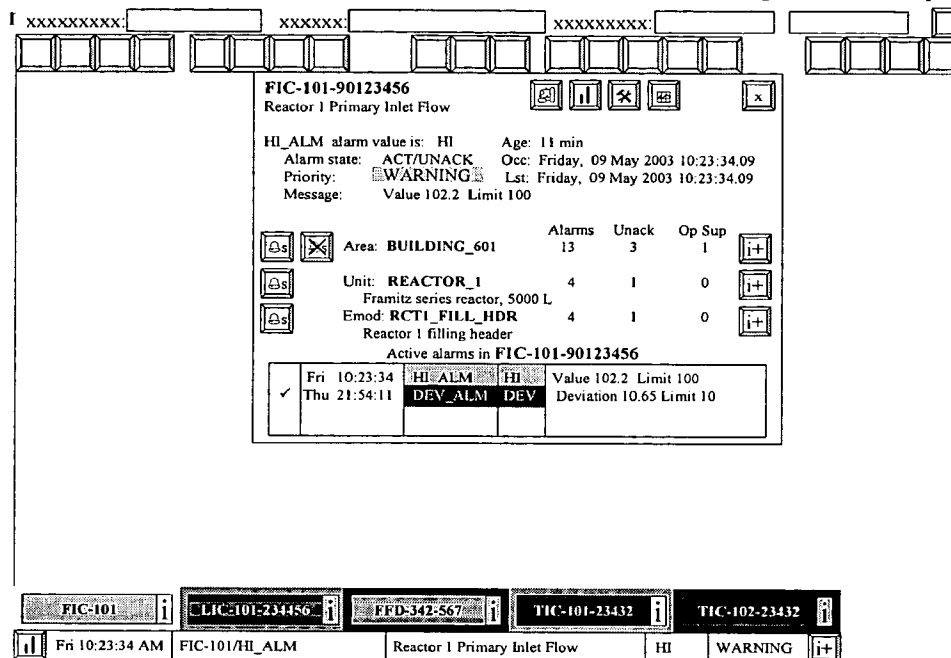
recently.

3. Clicking the 'i' button beside the button showing the new alarm, he sees a brief summary of the alarm that just occurred in the alarm banner. The operator doesn't remember seeing this alarm before, so he elects to view the extended alarm information display for the selected alarm by pressing the 'i+' button.

The display logic for the 'i+' button captures the module name ("FIC-101") for the alarm currently selected in the alarm banner, constructs a calling info string of "Display='DvAlarmInfo'; Module='FIC-101'", then passes it to workspace function "OPEN_DISPLAY".

The "DvAlarmInfo" display was configured with a panel category of "ALARMINFO". In the current framework, there is a (floating) panel configured to be an "ALARMINFO" category target, so that floating panel is chosen for the "DvAlarmInfo" display. The display currently open in that floating panel (if any) is closed, and an instance of the DvAlarmInfo display is opened there.

4. The display logic in the "DvAlarmInfo" display expects a module name to be in its launch information. Finding "FIC-101", it uses that name in calls to the data services layers to obtain information about FIC-101 and its containing Unit and Equipment



5. Now confident he understands the alarm situation in FIC-101, and its containing modules, he closes the "DvAlarmInfo" floating panel, and decides to look at the primary control display for FIC-101; by pressing the button so labeled in the alarm banner.

The display logic for the alarm banner buttons capture the module name ("FIC-101") and constructs a calling info string of "Panel='MAIN'; Module='FIC-101'; Select='FIC-101'; KeepARScrollOneDim", then passes it to workspace function "OPEN_PCD".

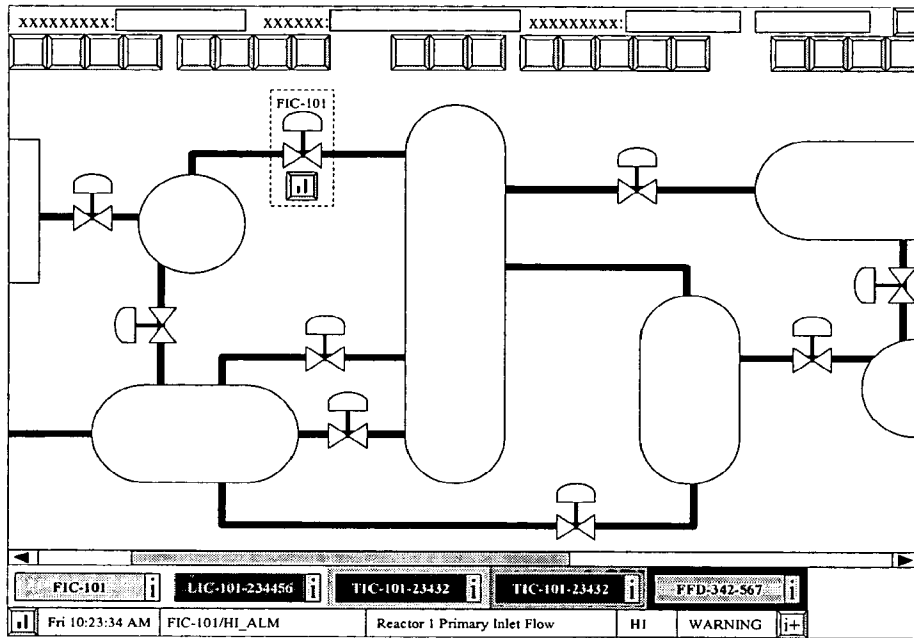
The OPEN_PCD function resolves the primary control display name "REACTOR1_TOP" for module "FIC-101". It then asks the workspace to resolve PANEL='MAIN', and to replace the display currently in that panel with REACTOR1_TOP.

REACTOR1_TOP originated through an import of a P&ID drawing from another system, so its native aspect ratio is much wider than the MAIN panel in the current framework. The "KeepARScrollOneDim" directive says that the aspect ratio for REACTOR1_TOP should be maintained while scaling it to fill the MAIN panel, with scroll bars for the dimension that won't fit.

The Select="FIC-101" directive is forwarded to "REACTOR1_TOP" telling it to resolve the "best" selectable graphic object associated with "FIC-101" and automatically give it selection focus (scrolling the display as necessary so the selected object is visible and as centered in the MAIN panel as possible.)

The presence of the "KeepARScrollOneDim" and "Select" directives overrides the default workspace behavior which remembers the scaling and scroll position last used on a display, for when it is opened again in the same user/session.

6. After looking at "near by" alarm conditions and process measurements, the operator chooses to make an adjustment to the setpoint on FIC-101, and watch how that control loop reacts. The faceplate display is the ideal interface for what the operator has in mind, so he pushes the faceplate button in REACTOR1_TOP for this module.



(Or since FIC-101 is still in the alarm banner, he could have pressed the faceplate button there.)

The display logic for the faceplate button captures the module name ("FIC-101") associated module, constructs a calling info string of "Module=FIC-101", then passes it to workspace function "OPEN_FPD".

The OPEN_FPD function resolves the faceplate display name "PID_LOOP_FP" for module "FIC-101". The "PID_LOOP_FP" display was configured with a panel category of "FP". In the current framework, there are two floating panels configured to be an "FP" targets, both are currently empty, so floating panel on the left is chosen as it was placed ahead of the other floating panel in the floating panel "use order" configuration. An instance of the PID_LOOP_FP display is opened there, passing it the launch information: "Module='FIC-101'".

The display logic in the "PID_LOOP_FP" display expects a module name to be in it's launch information. Finding "FIC-101", it uses that name in calls to the data services layers identify the parameters in FIC-101 it will be reading.

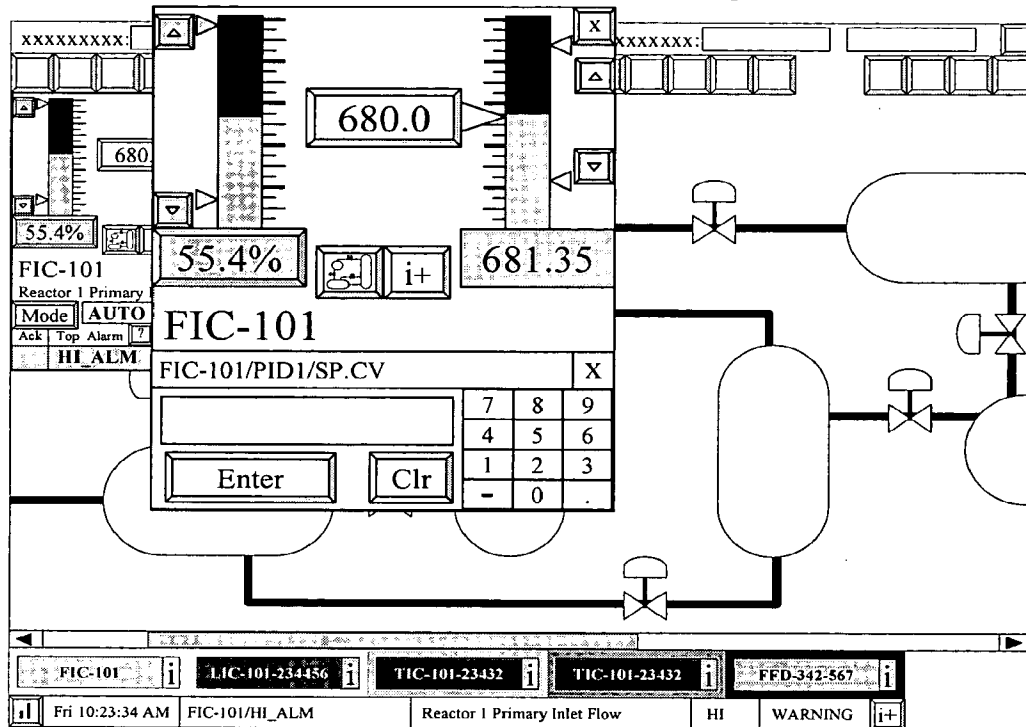
Several parameter/field values from FIC-101 are used repeatedly in the FIC-101 display (notably the scaling parameter associated with the PV and SP parameters) and for any update cycle of the display it is desirable that the scaling values be used

consistently. The "pre-update" logic for "PID_LOOP_FP" read the EU0 and EU100 values, engineering units string and decimal places information and stores them in "local display variables" which can be referenced by any of the graphic elements in "PID_LOOP_FP".

In short order, a new instance of "PID_LOOP_FP" appears in the floating panel initially located at its anchor point.

7. The operator thinks a significant SP change is appropriate (using the nudge up or down buttons won't do), so pushes on the button indicating the set point value.

The display logic for the SP button click is to ask the workspace to provide a standard numeric data dialog. The "PID_LOOP_FP" display is designed to also be used in



workspaces running on PDAs, so it constructs a parameter info string of "InParentDisplay; DockBottom; Title='FIC-101/PID1/SP.CV'" and passes it to the workspace function "NumericDataEntry".

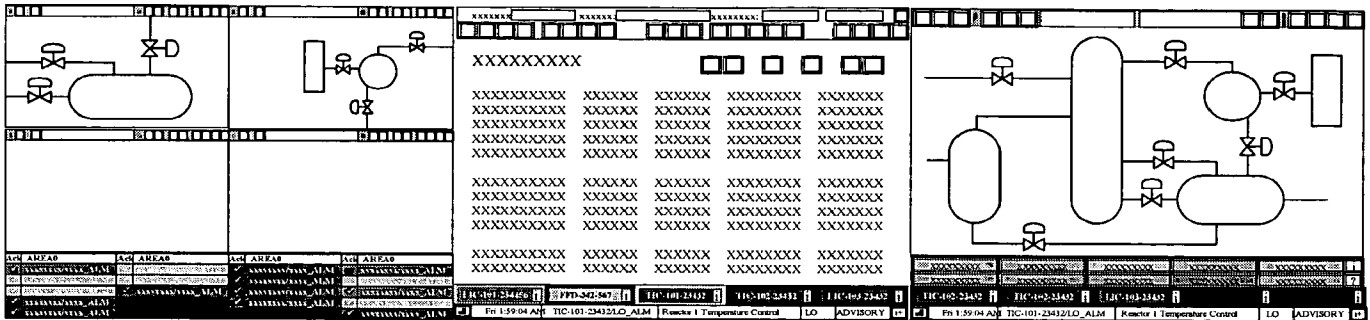
The NumericDataEntry workspace function sees that the workspace was launched with a "ShowKBOnScreen" preference (perhaps running on a hardware where the keyboard is not always present), so it chooses an instance of the standard numeric data dialog with an on screen keypad. The workspace resolves the dimensions and location of this instance of the PID_LOOP_FP display, and locates the dialog box at the bottom of the faceplate display.

8. Operator enters a new value for the setpoint, closes the on screen numeric pad. Within a second or two he sees the new value for setpoint reflected in the value shown on the setpoint button, knowing that the controller is now using/reporting the new value. The mode is in AUTO so he confirms some changes in the control (valve)

output, and shortly thereafter the PV starts moving in the desired direction. Satisfied with the change to this setpoint, he closes the faceplate, and monitors the nearby process measurements on the primary control display.

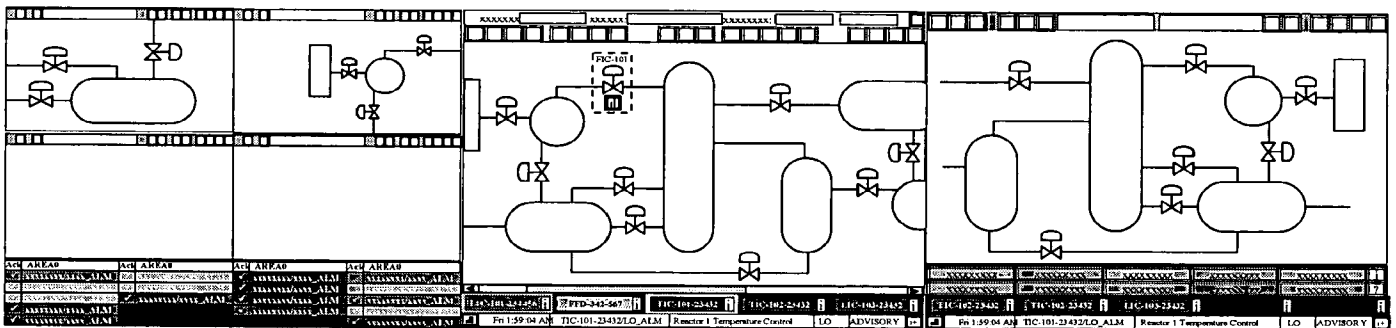
1.7.1.1.2 3-Monitor Workstation Environment

1. The system has been pretty quiet since the operator came on shift. Taking advantage of the 3 monitor workspace, he's been keeping an eye on 3 different parts of the plant that where he's made recent changes, and comparing the production totals so far for the day with target values.



2. A new alarm appears in the alarm banner. The operator recognizes the tag and confirms the module description: his last setpoint change was supposed to fix this, but apparently it hasn't. He's familiar with this unit, and goes directly to the primary control display by pushing the alarm banner button with the module's name.

(Like the single monitor scenario, the primary control display appears on the MAIN panel, with focus on the graphic element representing the module.)



3. The operator now believes the problem is really upstream of the module that is reporting the alarm. He sees there is more to this process display out of view to the left, so he scrolls to bring it into view. He pushes the faceplate button for an upstream control loop makes a change, verifies the control action starting to take effect.

He knows he's going to have to check back on this a few times in the next hour, so he decides to put a copy of this display in one of the empty panes in the left monitor:

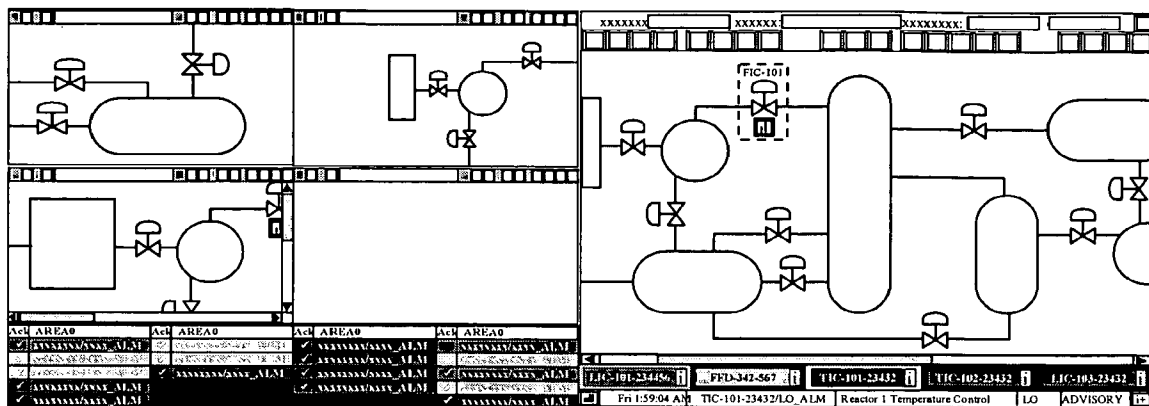
- He pushes the "Copy panel content" button in the toolbar over the MAIN panel. The display logic behind the button prepares a parameter information string of "Panel=MAIN" and calls the workspace function CopyPanelContent. The Copy PanelContent function captures the display name currently in the

specified panel, the launch information used to create that display, and the current scaling, and scroll position settings.

- He then pushes the "Paste" button in the information and tool button bar of an empty panel on the left monitor.

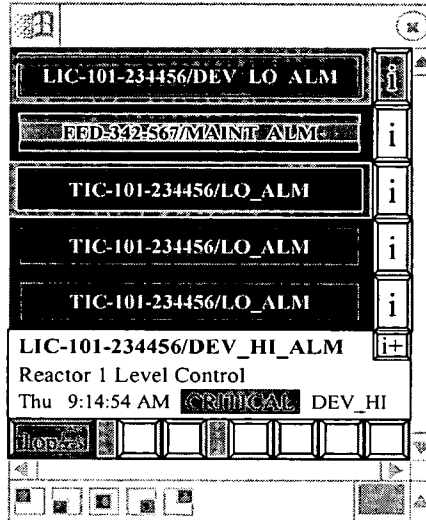
The Paste button in framework panels essentially prepares a parameter information string of "Panel='<my panel id>'; UseSourceScale", and calls the workspace function that "paste copied panel contents" to "this" panel.

The a new instance of the display, with the original launch information is opened in the panel. The scaling of the source display is preserved, but since the panel is half the size of the source panel, the view is centered on the center point of the source view, and horizontal and vertical scroll bars appear.

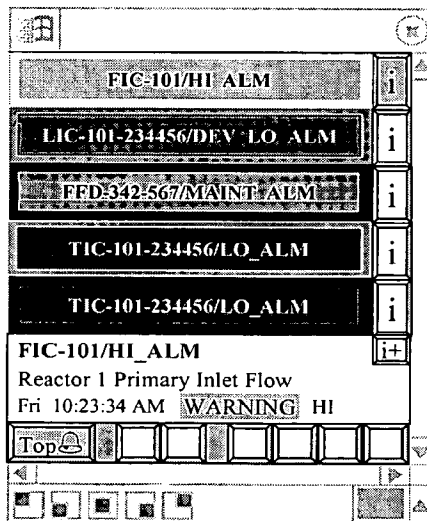


1.7.1.1.3 Pocket PC Environment

1. The system has been pretty quiet so far today. The operations supervisor keeps an eye on things as he moves about (in his office, talking with the operators on shift, meeting with the production planning team) using his PDA. While they switch over to making a new product grade, he likes to keep the "TOP_ALARMS" display open in the MAIN panel. (He can always get back to this display by hitting the "Top" button in the TOOLBAR panel, always at the bottom of the display area.



2. While in the break room talking to the new operator, his PDA makes the WARNING-level alarm sound. He looks at his PDA and (judging by the white background) one new alarm has just occurred.

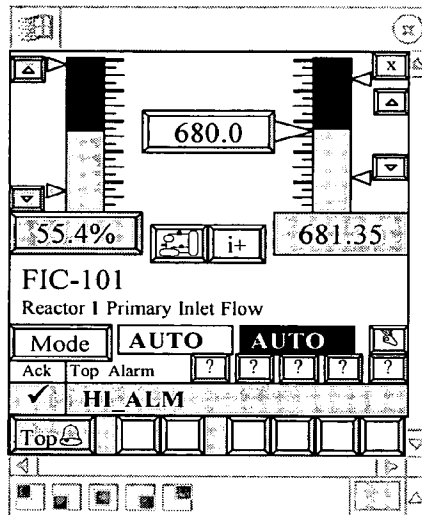


The ops supervisor knows what should happen when responding to this alarm, but his crew in the control room has the ball. Since this can be tricky, he pushes the "i+" button to check for other alarms in this module, equipment module, and unit.

The display logic for the "i+" button is designed to call up the ALARMINFO display for the selected module. Normally the ALARMINFO display would be retrieved from the DEFAULT subtree under the display configuration storage root directory. However, this workspace was started with the launch information "DisplayPref=PDA", so it will attempt to find a display Definition named ALARMINFO in subtree named PDA, before looking for it in the DEFAULT subtree. It turns out that a version of the ALARMINFO display better suited to the display size of this handheld device was found under the PDA tree, so it was automatically substituted for the default ALARMINFO display.

Not seeing anything in the equipment module or unit that indicates he should step in, he opens the faceplate display for FIC-101, by pushing the faceplate button in the ALARMINFO display.

Shortly after opening the faceplate display for FIC-101, he sees that his crew has already acknowledged the new alarm. A bit later he sees a setpoint adjustment (just what he would have done.) Confident that the time he's spent in training this crew is paying off, he can get back to his conversation in the break room.



Upon activation, that instance of DeltaVRtObjectBrowser parses the launch information, and finding no panel designation, negotiates with the workspace to determine the panel it should use, location/dimensions, etc. In this case, since no other workspace configuration information was applicable to this application, the MAIN panel is selected (it had been configured as the default destination panel in this workspace framework). The workspace deactivates the process graphics display currently in the MAIN panel, and tells the DeltaVRtObjectBrowser that it now has control of that panel.

The 'DeltaVRtObjectBrowser' application chooses a layout suitable for the dimensions of the panel it's been given. In this case, having a large panel available, it chooses a layout that occupies the full height of the panel, but since it can't use the full width effectively, it uses the right half of the available panel space.

XXXXXXXXXX:		XXXXXXXXXX:	XXXXXXXXXX:	
<div style="display: flex; justify-content: space-between;"> <div style="width: 45%;"> <p>VSPMP-101</p> <p>Location: Building 6, Level 2, Grid A4</p> <p>ID Tag: 0023-245</p> <p>Speed limiter set to 1800 RPM</p> <p>Auto stop will trip below 100 RPM</p> <p>Maximum flow (limited): 2400 GPM</p> <p>Last Entry: Tues Mar 14 2003 4:16 PM</p> <p>Operations Journal</p> <p>Asset Management System</p> <p>Class: ACME XL3306 Centrifugal Pump, 6-inch outlet</p> <p>Flow capacity: 100-2800 GPM</p> <p>Working speed range: 60-2000 RPM</p> <p>Manufacturer Operating Guidelines</p> <p>Drawings and Pictures</p> <p>Class: Type B Variable Speed Pump</p> <p>Variable Speed Pump, manual two-stage start sequence</p> <p>Operating Procedures</p> <p>Maintenance Procedures and Contacts</p> <p>Is in Plant Area: BUILDING_601</p> <p>Is a part of: FIC-101 Reactor 1 Primary Inlet Flow Control</p> <p>Includes:</p> <p>Upstream: BV-101-23 Reactor 1 Inlet Block Valve</p> <p>Downstream: SSV-101-1 Reactor 1 Primary Inlet Valve</p> </div> <div style="width: 5%; text-align: center;">x</div> </div>				
<div style="display: flex; justify-content: space-between;"> <div style="width: 45%;"> <p>FIC-101</p> <p>LIC-101-234456</p> <p>TIC-101-23432</p> <p>TIC-101-23432</p> <p>FFD-342-567</p> </div> </div>				
<div style="display: flex; justify-content: space-between;"> <div style="width: 45%;"> <p>Fri 10:23:34 AM</p> <p>FIC-101/HI_ALM</p> <p>Reactor 1 Primary Inlet Flow</p> <p>HI</p> <p>WARNING</p> </div> <div style="width: 5%; text-align: center;">i+</div> </div>				

3. He reviews the information about VSPMP-101:

- The first section has information about this specific pump. The configuration guys have been good about putting location and physical ID information in the "extended description" for most of physical plant objects; which has come in very handy when they have to find them for physical inspections.

A button is available here to open the "Operations Journal" application, for VSPMP-101. In this case it indicates that there is at least one entry in the OJ for this device, but it was made some time ago.

Since AMS was purchased for this system, and VSPMP-101 is a recognized asset

identifier in the AMS database, a button to call up the AMS asset viewer application also appears.

- The second section contains information about this type of pump (there are lots of these on site). In this case, there are two levels of equipment-type (class) information:
 - The more specific class information is about the make and model of pump. The configuration guys were nice enough to put in links to the manufacturer's operating guidelines documentation...and someone was nice enough to take some identification pictures.
 - The more general class information has links to the site training documents; standard procedures for starting/stopping our "Type B" variable speed pumps (regardless of manufacturer, model, or capacity)¹⁸. Also, the Maintenance staff has apparently finished the "first line troubleshooting" documents for this kind of equipment.
- The third section is reference information where this pump (and its control module) fits into the control system (hierarchy). The lists and buttons let you move the 'DeltaVRtObjectBrowser' up and down through the control hierarchy.
- The fourth section lets you move the 'DeltaVRtObjectBrowser' to upstream or downstream objects.

The operator sees the speed and flow capacity limits for this pump and wonders if they are pushing the limits. He pushes the faceplate button in the 'DeltaVRtObjectBrowser' application and the faceplate for VSPMP-101 appears in the left floating panel. It looks like the current pump speed is well under the limit indicated in the description for this valve. Hmmm...

4. The operator decides to look downstream of the pump, and pushes the view button on the single downstream object listed: SSV-101-1 Reactor 1 Primary Inlet Valve. The browser shifts focus to this control valve. The operator notices a relatively new entry associated with this piece of equipment in the Operations Journal, so he pushes the button to call it up.
5. The Operations Journal application opens in the MAIN panel, replacing the Object Browser. The most recent entry for SSV-101-1 indicates that on the last physical inspection, the upper limit switch appears to be loose or damaged. (The field tech doing the inspection included a photo of the problem she saw.) She said the valve appeared to working OK at the mid-range travel position, but feared that it might not work correctly at nearly full open. Repair parts are on order...expecting repairs to be completed on Friday.

There's the problem! The valve isn't allowing the expected flow rate at our new higher production rate. Better leave my own entry in the Operations Journal:

- The operator hit the "last display" button in the TOOLBAR panel over the MAIN panel. The Operations Journal application was replaced by the Object Browser application; still with focus on SSV-101-1.
- Since the control valve was part of the FIC-101 equipment module, he just hits the view button to move up the control hierarchy.

¹⁸ Also a good place to access the training documentation on how the faceplates were designed to interact with this class of equipment.

- With focus on FIC-101, he hit the Operations Journal button to bring up that application, looking at the journal entries for equipment module FIC-101.
- The operator adds his own entry, saying that he is seeing deviation alarms when trying to achieve flow rates of 2200 GPM or more, and confirms that the limit switch problem found on SSV-101-1 (making a Operations Journal link to the other object) seems to be causing the deviation alarm problem at the current production rates.

He knew that his supervisor would be reviewing the recent Operations Journal entries, and would let the production planning team know that they shouldn't count on maximum capacity of this product grade for the next few days.

1.7.1.3 Monitoring Multiple "Hot Issues"

1.7.1.3.1 Single Monitor Workstation Environment

1. It's been one of those days! Ever since coming on shift, it's been one thing after another and some of the operator's changes haven't had the expected results!

The operator had just finished responding to an alarm by making a setpoint change. Satisfied the change was accepted by the controller, he'd like to watch the primary control display for a while to make sure things were going in the right direction this time! But a few more alarms are pending that need attention too...so...good time to define a "display alert" to watch things while his attention is elsewhere.

The temperature setpoint change he just made is going to take a good hour to raise the temperature on the product in Reactor 1 to the new target. This should be a no-risk change, but they'll lose this production run if they don't achieve the temperature target for some reason.

He clicks on the "Add Display Alert" button in the TOOLBAR panel over the MAIN panel showing the control display for Reactor 1. The pointer changes to the "value picker" shape and he moves it over the digital temperature measurement value that he wants to set the display alert on, and clicks it. The Display Alert dialog appears.

2. He decides a "Target Alert" is what he wants, so he leaves that tab selected. The parameter that he wants to set the alert on is already filled in (thanks to the value picker). He decides to enable an initial delay of 1 hour, before checking that the target value had been reached. He enters the target value (720 degrees) and an acceptable deviation band (+/- 5 degrees). He sets the alert check duration of 1 hour (making sure the temperature doesn't drop or overshoot for at least an hour after the target is achieved). Since he doesn't have anything more to do when this alert is removed, he clears the "Ack before removing" option.

Target Alert

After 1 hours and 0 minutes (initial delay)

TI-101-2/A11/IN.CV = 720

within a deviation of 5

For a duration of 1 hours and 0 minutes, then alert is removed.

☐ Acknowledge before removing

Comment:

Add Display Alert

FIC-101 LIC-101-234456 TIC-101-23432 TIC-101-23432 FFD-342-567

Fri 10:23:34 AM FIC-101/HI_ALM Reactor 1 Primary Inlet Flow HI WARNING

Everything looks fine, so he hits the "Add Display Alert" button. The dialog closes and a runtime workspace starts add this new display alert. An hour from now, it will start checking that the value for TI/101-2/A11/IN.CV is 720 (+/- 5) degrees, and stays there for the next hour. After that point, the display alert will automatically remove itself.

3. According to the production plan for the day, it's time the operator changed the blending unit over to making their premium product. After he completes the necessary control parameter changes, he recalls that his supervisor said they might have some intermittent shortages of Hydrogenicine over the next few days. And he remembers from last time that the high and low alarm limits are way too loose for that flow stream when running this product grade.

Having better things to do than watch that flow level for the rest of his shift, he calls up the display showing the blending flow streams, and "Adds Display Alert" to the Hydrogenicine stream's flow measurement.

The dialog appears, but this time he switches to the Range Alert tab. He consults the

grade formula reference documents and confirms a flow rate of 112 GPM for the Hydrogenicine stream, so he sets pretty tight upper and low range limits for the display alert, so he will learn much earlier if feedstock shortages are causing problems maintain the necessary flow level for this product.

4. He sees that the flow rate of 112.2 has already been established so there is no need for an initial delay on the display alert. Also, the plan is to be making this product until at least the end of his shift, so he clears the alert "duration" time; letting the alert check be active until he manually removes it. He's done defining this alert.

Target Alert Range Alert Ramp Alert

☐ After hours and minutes (initial delay)

stays between: ...

within a deviation of ...

☐ For a duration of hours and minutes, then alert is removed.

☒ Acknowledge before removing

Comment:

Add Display Alert

5. The operator also sees on the production plan that he needs to move some finished product from intermediate storage to the distribution storage tanks. These are BIG tanks (relative to the piping) so these operations take a long time and they didn't invest in automatic controls.

The operator pulls up the tank farm process display, and uses the Object Browser Application to get the link to the product movement procedure checklist. After manually opening and closing the appropriate block valves, he starts the pump and verifies a steady flow measurement as the product is transferred. The production plan is to achieve a level of 360 inches in the destination tank, and the plan calculated that at the target flow rate, that level will be achieved in 12 hours.

With no discharges planned, operator expects a steady increase of the level of the tank from its present measurement, to the target over the next 12 hours. Rather than set a Target Alert (with no checking going on for 12 hours) he chooses a Ramp Alert, which will be checking for a steady "ramped" measurement throughout the next 12 hours.

Since the next shift operator will need to shut off the transfer pump and close valves, he checks "Acknowledge before removing" so the completed alert will get the next operator's attention. He also adds a comment to remind them what needs to be done. He's done setting up this display alert.

Target Alert Range Alert Ramp Alert

☐ After hours and minutes (initial delay)

ramps to ...

within a deviation of

☒ Ramp duration of hours and minutes, then alert is removed.

☒ Acknowledge before removing

Comment:

6. To check his Display Alerts he's set up, the operator presses the "Display Alerts Status" button in the ALARMBANNER panel. This button replaces the content of the MAIN panel with the Display Alerts Status application.

7. He sees the three display alerts he's set up...the alert rules look OK. He notices that

Display Alerts:	Off target:	Was off target:	On target:	All:	
Initial delay:				1	<input type="button" value="Acknowledge all"/>
Running:	0	0	2	2	<input type="button" value="Remove all"/>
Complete:	0	0	0	0	
Unacked:	0	0	0	0	<input type="button" value="Show details"/>

<input type="text" value="112.81"/>	<input type="button" value="FIC-B1-HGC/PID1/PV.CV stays between 110 and 115"/>	<input type="button" value="X"/>
<input type="text" value="129.5"/>	<input type="button" value="LI-TF1-PRD23/A11/IN.CV ramping to 360 [+/- 4] by 23:55"/>	<input type="button" value="X"/>
<input type="text" value="684.34"/>	<input type="button" value="TI-101-2/A11/IN.CV = 720 [+/- 5] by 13:42 until 14:42"/>	<input type="button" value="X"/>

LIC-101-234456 FFD-342-567 TIC-101-23432 TIC-102-23432 LIC-103-23432

Fri 1:59:04 AM TIC-101-23432/LO_ALM Reactor 1 Temperature Control LO ADVISORY

two of them are running already (since he didn't configure a initial delay on them) and they are reading values which are currently on target.

8. Sometime later, as his attention is on another section of the plant, the operator sees the

Display Alerts:	Off target:	Was off target:	On target:	All:	
Initial delay:				0	<input type="button" value="Acknowledge all"/>
Running:	1	0	2	3	<input type="button" value="Remove all"/>
Complete:	0	0	0	0	
Unacked:	1	0	0	1	<input type="button" value="Show details"/>

<input type="checkbox"/>	<input type="text" value="109.66"/>	<input type="button" value="FIC-B1-HGC/PID1/PV.CV stays between 110 and 115"/>	<input type="button" value="X"/>
	<input type="text" value="208.21"/>	<input type="button" value="LI-TF1-PRD23/A11/IN.CV ramping to 360 [+/- 4] by 23:55"/>	<input type="button" value="X"/>
	<input type="text" value="719.34"/>	<input type="button" value="TI-101-2/A11/IN.CV = 720 [+/- 5] by 13:42 until 14:42"/>	<input type="button" value="X"/>

9. He sees all three of his display alerts are running now, but the flow value on the Hydrogenicene stream is not staying in the range he needs it to be in. It is the one showing un "unacked" status (and is sorted to the top of the list), so he acknowledges it, and presses the button to bring up the primary control display for that control module to make some compensating control parameter changes. Later, he comes back to the Display Alerts Status display, and pushes the "Change Display Alert" button for this display alert. It's done it's job, so he uses the Change Display Alert dialog to remove it.
10. During the next shift, the new operator pushes the Display Alerts Status display button to review what the previous shift had left him. He sees a single display alert left over, so he presses the Show Details button to get the full scoop on it:

Display Alerts:	Off target:	Was off target:	On target:	All:	
Initial delay:				0	<input type="button" value="Acknowledge all"/>
Running:	0	0	1	1	
Complete	0	0	0	0	<input type="button" value="Remove all"/>
Unacked:	0	0	0	0	<input type="button" value="Show details"/>

<div style="border: 1px solid black; padding: 2px;">315.17</div>	LI-TF1-PRD23/A11/IN.CV ramping to 360 [+/- 4] by 23:55 Per DPP item 21: When target tank level reached, stop transfer, reset valves. Product Storage Tank 23 Level Display alert set at 12:42 by RIDLEY TANK FARM:
------------------------------------------------------------------	---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

11. The operator thinks "Good ol' Ridley lookin' out for both of us!". He checks the Daily Production Plan and sure enough all the details for the product transfer is listed under item 21. He's got a couple of hours before he needs to come back to this, so he goes on about his business.
12. After a couple of hours, the operator notices the Display alert indicator in the alarm banner area had turned white and began flashing. He opens the Display Alert Status display, and sees that the display alert indicates it's completed, and needs acknowledgment.

He acknowledges the completed display alert (which will remove itself shortly). He presses the button for the primary control display for LI-TF1-PRD23 so that he can stop the transfer pump and reset the transfer valves.

Display Name:	On target	Not on target	On target	Not on target	Count	
Initial delay:					0	<input type="button" value="Acknowledge all"/>
Running:	0	0	0	0	0	
Complete:	0	0	1	1		<input type="button" value="Remove all"/>
Unacked:	0	0	1	1		<input type="button" value="Show details"/>

<input type="checkbox"/>	358.44	<input type="button" value="E"/>	LI-TF1-PRD23/A11/IN.CV ramping to 360 [+/- 4] by 23:55 Per DPP item 21: When target tank level reached, stop transfer, reset valves. Product Storage Tank 23 Level Display alert set at 12:42 by RIDLEY TANK FARM:	<input type="button" value="X"/>
--------------------------	--------	----------------------------------	---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	----------------------------------

1.7.1.4 Responding to an "Alarm Flood"

1.7.1.4.1 Single Monitor Workstation Environment

- Things had been going pretty well, but a big lightning storm blew through and the new alarms are flying through the alarm banner and Event Chronicle faster than the operator can keep up with. He pushes the button in the TOOLBAR panel for the "Alarm Profiles" application. It appears in the MAIN panel replacing the previous

Recent Alarm Profiles Alarms in previous 1 hr

THISUSER	Alarms:	25	25
In chart:	26	25	

Active Unack Top AREAS:

AREA_A	Alarms:	18	18
In chart:	17	16	

AREA_D	Alarms:	1	1
In chart:	4	1	

AREA_G	Alarms:	0	0
In chart:	2	0	

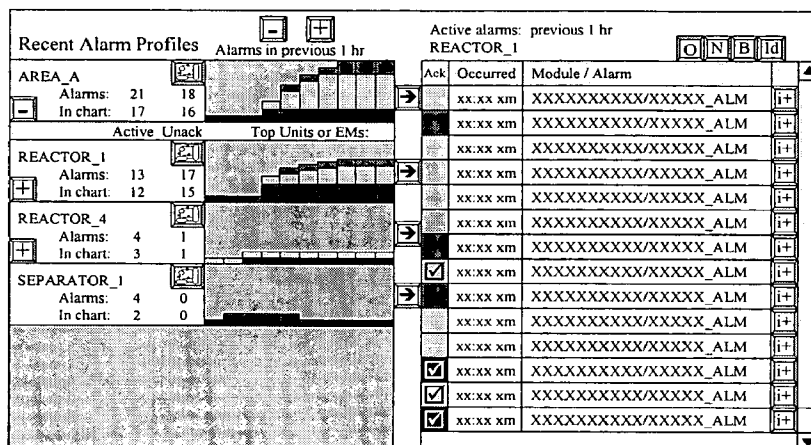
AREA_E	Alarms:	0	0
In chart:	1	0	

Fri 1:59:04 AM TIC-101-23432/LO_ALM Reactor 1 Temperature Control LO ADVISORY

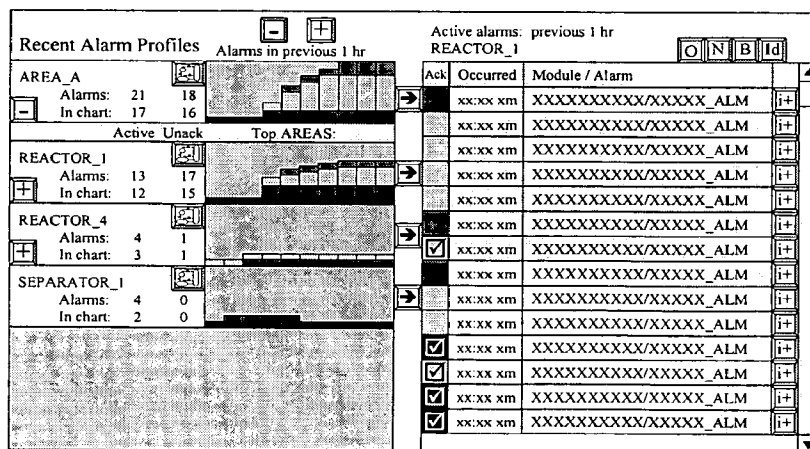
display.

- The alarm profile charts indicate active alarm counts, stacked by priority, and charted over the previous hour. Just as he thought, there had been a lot of new alarms; all pretty much in the last 45 minutes. The topmost chart shows the active alarm profile for all (active) areas under his control (alarm management scope), and it automatically shows charts for each of the five most active plant areas. He quickly sees that nearly all the new alarm occurrences are coming from AREA_A, so he presses the "Expand" button for that area to drill in for more detailed charts.

The Alarm Profiles application changes to focus on AREA_A, which becomes the topmost chart, with the charts for the 5 most active units/equipment modules in the plant area. Again it shows that nearly all the new alarms are coming from the REACTOR_1 unit. He pushes the "List alarms" button for REACTOR_1, and a list of all active alarms associated with that unit, that occurred within profile time window (the previous hour) appears in the right side of the display.



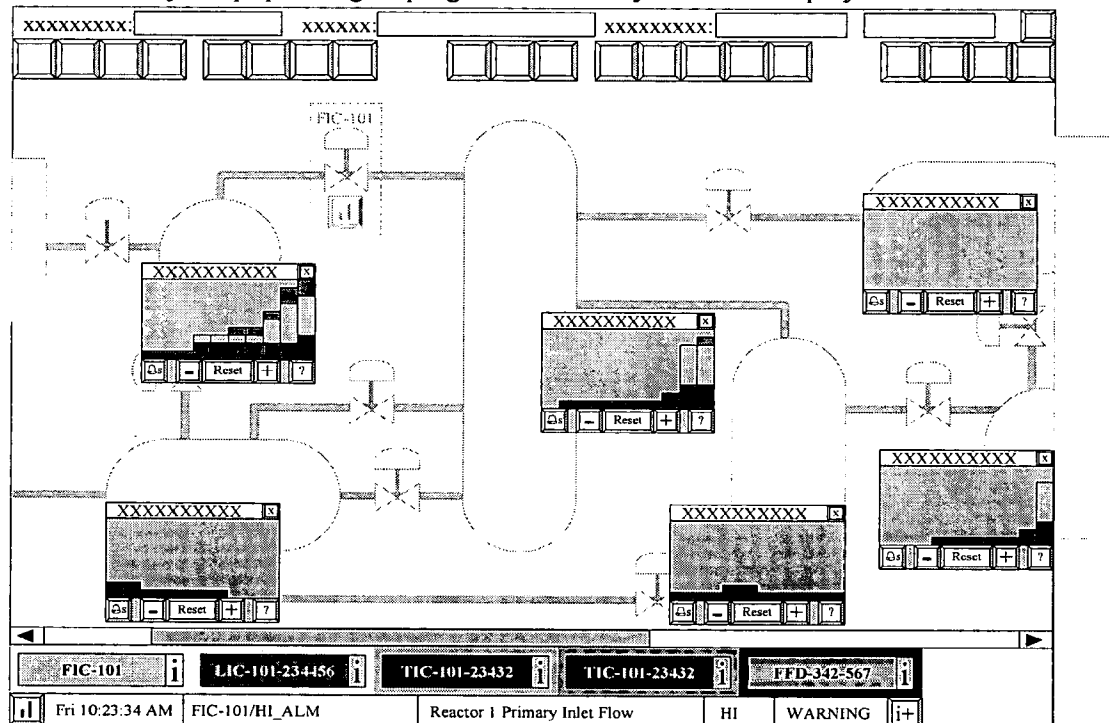
- Too many alarms for them all to show. Hmmmm...it looks like the leading edge of the alarm burst occurred about 45 minutes ago (the 4th bar in the chart for REACTOR_1) so he clicks on that bar in the chart, and the alarms that occurred during that time interval are highlighted in the alarm list:



4. Now he's closing in. Using the "i+" buttons, he pops open the DvAlarmInfo display for each of the highlighted alarms to get the full details. It's looking like a couple of I/O failures (right after that big lightning strike?) tripped the first alarms, then a couple of deviation alarms came soon after that. Now looking at the newer alarms in the list, he's beginning to suspect that most of them were caused indirectly from those I/O failures, and the control actions the system then took to abort an SFC that was in progress.

To get a another view on the alarm profile for REACTOR_1, he presses the "primary control display" button for it. Yup, quite a few alarm indicators on this display. He pushes the "Advanced Display Features" button on the TOOLBAR panel, and selects "Display Layers". A dialog appears showing the layers configured for the current display (in the MAIN panel) and their visibility status. He doesn't see any layers in the display that aren't already being displayed (sometimes the display guys will put in a specific layer to show alarm information...but not this time.)

So he goes back to the "Advanced Display Features" dialog and now selects "Add Alarm Profiles". This causes the runtime workspace to find the graphic elements associated with unit and equipment modules, their location on the screen, and creates a temporary display layer for the existing display which shows active alarm profiles for each major equipment grouping. The other layers in the display are subdued



(semi-transparent) to make the alarm profiles easier to see.

5. The operator quickly sees the parts of the REACTOR_1 unit which have been quiet (alarm-wise) over the last hour, so he can stop worrying about those. It also appears that the burst of alarms in the downstream EMs in the unit followed the burst of alarms in the upstream EMs. The operator is gaining confidence that the alarm flood is due to a single chain reaction through the unit, and now he knows pretty clearly

Display Alerts:	Off target:	Was off target:	On target:	All:	
Initial delay:				0	<input type="button" value="Acknowledge all"/>
Running:	0	0	0	0	<input type="button" value="Remove all"/>
Complete	0	0	1	1	
Unacked:	0	0	1	1	<input type="button" value="Show details"/>

<input type="checkbox"/>	<input type="text" value="358.44"/>	<input type="button" value="G"/>	LI-TF1-PRD23/A11/IN.CV ramping to 360 [+/- 4] by 23:55 Per DPP item 21: When target tank level reached, stop transfer, reset valves. Product Storage Tank 23 Level Display alert set at 12:42 by RIDLEY TANK FARM:	<input type="button" value="X"/>
--------------------------	-------------------------------------	----------------------------------	---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	----------------------------------

where and when the chain reaction started.

He pushes the Alarm Summary button in the alarm profile for the most upstream EM in the chain, to start sorting out the secondary effects.

1.7.1.5 Using History Data

1.7.1.5.1 Single Monitor Workstation Environment

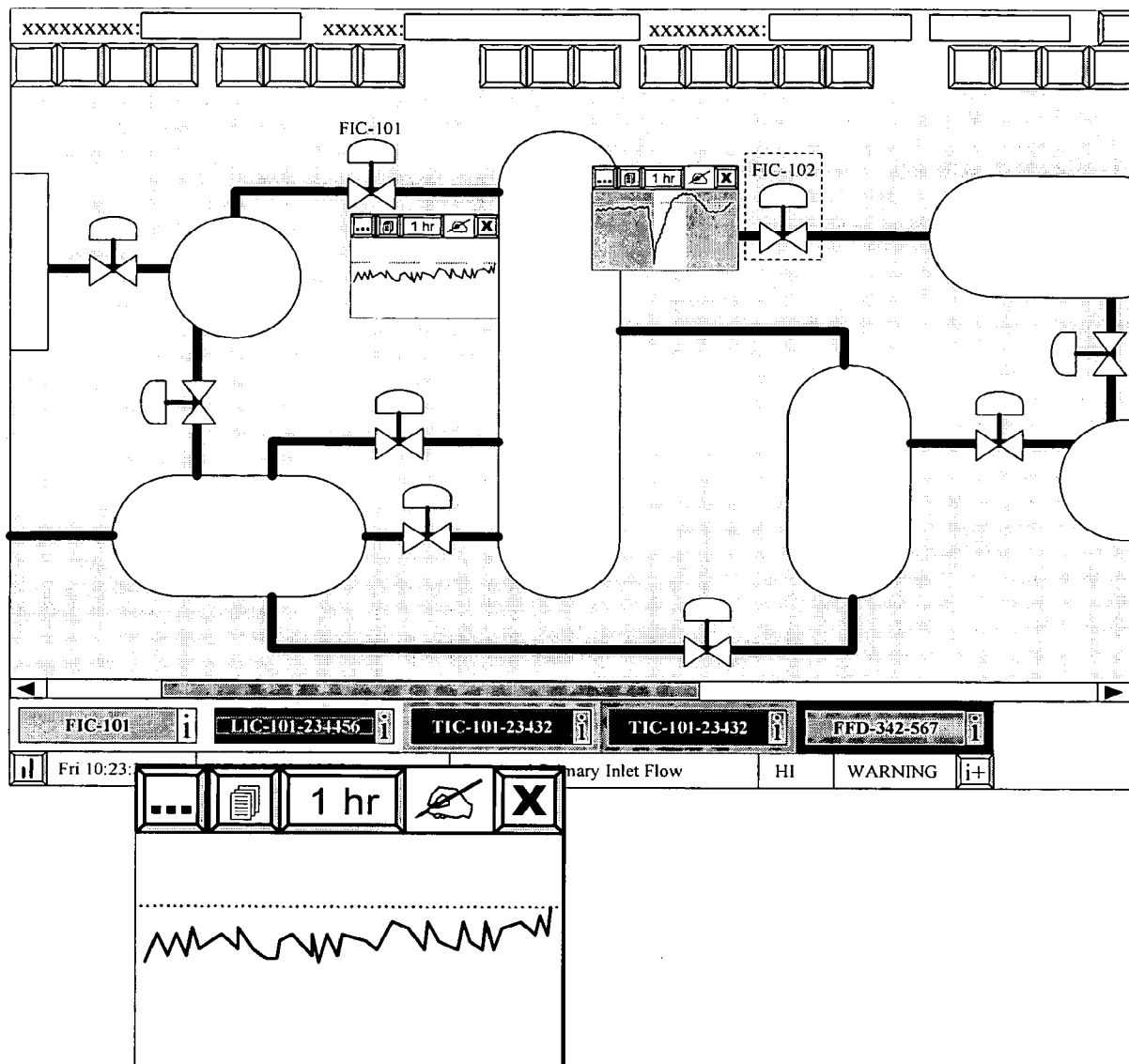
1. The operator had just gotten on shift, and got a quick rundown on what was going on from the previous shift before they hurried out the door for the "watering hole". Alarm-wise, the system was pretty quiet, but he noticed some hard to explain swings in some of the flow measurement surrounding REACTOR_1. "Time to get some 'historical perspective'", he thinks.

While he has the primary control display for REACTOR_1 in the MAIN panel, he presses the "Add Trend" button in the TOOLBAR panel. The pointer changes to the "value picker" shape and he moves it over one of the digital flow measurements that is in this display, and clicks it.

The workspace resolves the digital value clicked to be a data link to FIC-101/PID1/PV.CV. Since the maximum for trend pop-ups configured for this panel has not been exhausted, the workspace creates a trend pop-up for FIC-101/PID1/PV.CV immediately beneath the digital value clicked on.

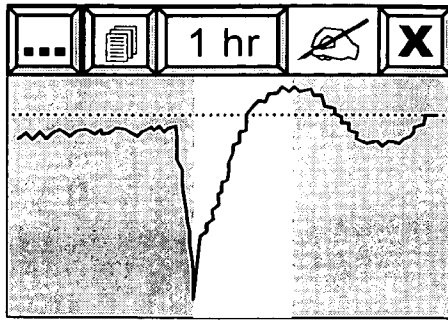
The new trend pop-up is logically attached to the current display in the MAIN panel; closing when the display goes away, reappearing when that display appears via "last/forward" display operations.

2. The operator sees the trend pop-up for the digital value he clicked on. He repeats this process to create trend pop-up for another flow measurement (in FIC-102) on the same display. He drags them bit so they don't cover other information on the display he wants to see:



Looking closely at the trend pop-up for the FIC-101 flow measurement He sees the history data for the last hour plotted. He knows trend pop-ups auto scale, but never auto scale to show more than 10% of the full EU range, and in this case, since the plotted values have scaled to a small band in the center of the plot, they can't have changed by more than 1-2% in the last hour. Nothing interesting here, so he hits the close button to dismiss this trend pop-up.

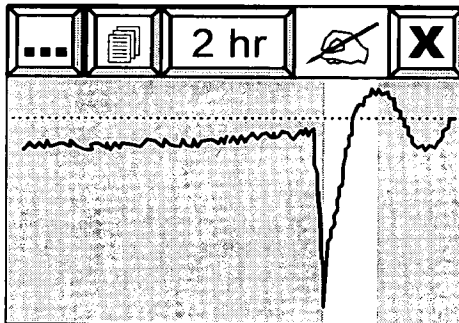
4. He turns his attention to the trend pop-up for the FIC-102 flow measurement:



He sees data for the last hour (the default configured for trend pop-ups for this panel) but this time the data occupies the full horizontal area, so he knows it's been changing by 10% of full scale (or more) over the last hour...Hmmm. He also sees:

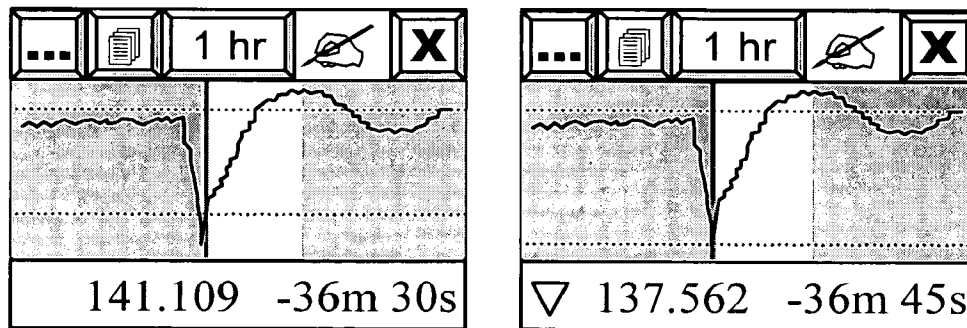
- The plotted data continues to update: the plot is in "real-time" mode (confirmed by the icon in the control bar).
- The background shading helps him quickly see the minimum and maximum values in the period plotted. (The configurator chose a warm color for the periods where the value was rising between first/last and min/max values; a cool color for when the value was falling.
- The horizontal reference dotted line for the current value helps him see that for most of the last hour, the values have been less than what they are now.

6. "That was quite a dip about a half hour ago!" To check the value for a longer time period he clicks on the period button in the control bar, and it cycles to a 2 hour view:



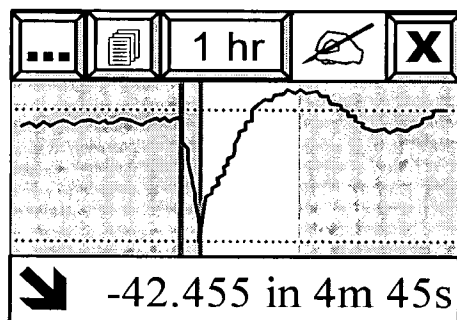
"No, nothing much interesting back there", as he clicked the the period button cycling to 4 hours and 8 hours, and then back to 1 hour.

7. "I wonder what that low reading was" as he clicked on the plot in the vicinity of the low point. The pop-up layout changes to shrink the plot area a bit, show a vertical plot cursor, and a legend bar indicating the value and relative time to the sample under the plot cursor:



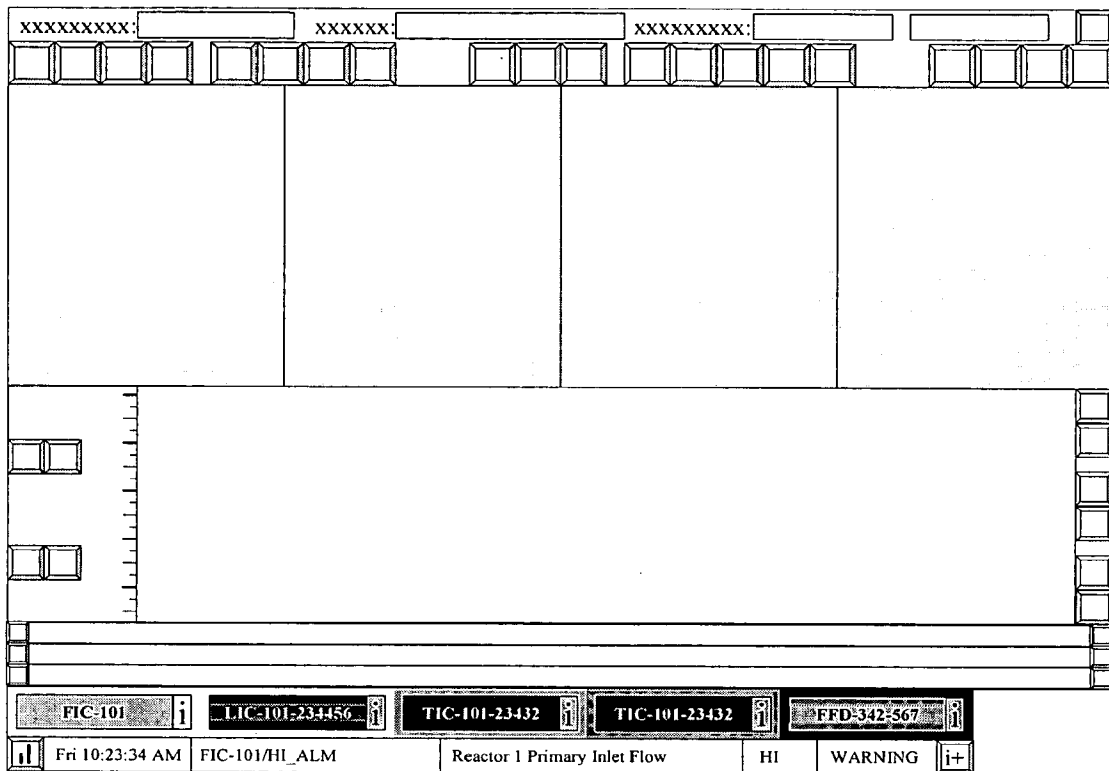
He can tell by the horizontal cursor value reference line that he didn't quite hit the minimum, so he uses the keyboard arrow keys to nudge¹⁹ the plot cursor back a couple of samples until he sees the "minimum value" icon appear in the legend bar.

8. "Ouch!", he thinks. "That was a pretty low flow value for a bit there...and it dropped fast!". How much, how fast? He lets the trend pop-up do the math: he right clicks on the sample just before the value dropped, and the reference cursor appears, and the legend bar changes to show the "two data points delta" representation: the change in value and time interval between the two cursors:

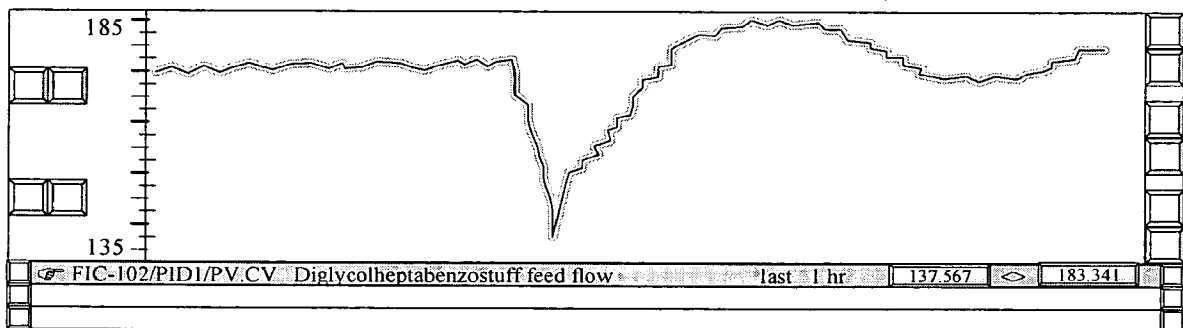


9. "Hmmpf! I wonder if the downstream flows saw this disturbance?" he thinks. So he scrolls the display in the MAIN panel to the left to show more of the downstream process equipment. He drops a few more trend pop-ups on some of the flow measurements in that part of the process, and in some he sees some unexpected patterns also.
10. "This ain't right! Time to build me a real trend display to keep on top of this." So he picks the menu on the TOOLBAR display and navigates down to the "Build New Trend Display" menu item. A new template trend display appears in the MAIN panel. The template has four faceplate sized (sub)panels on the top and a "History View" gadget (control) occupying the lower half of the display area:

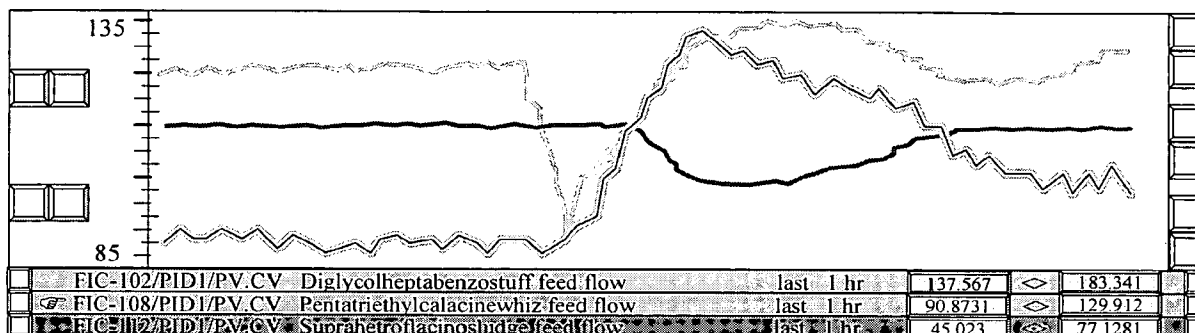
¹⁹ Or double clicking near the bottom of the plot area (or something else) to easily select the minimum value in range.



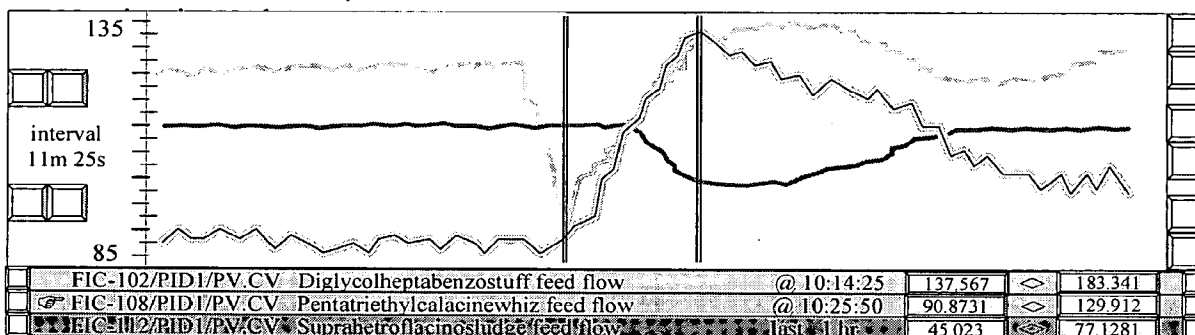
11. "Yup, that's what I want"...so he hits the "Back Display" button in the TOOLBAR, and the previous process graphic reappears in the MAIN panel (complete with the trend pop-ups that he had created on that display). He hits the "COPY" button on one of the trend pop-ups that shows the flow disturbance, then "Forward Display" in the TOOLBAR and then the "PASTE" button in the History View control. The control shows a larger trend trace for the flow value he'd copied; all set up to show the same period (last hour) and scale as he had copied from the trend pop-up:



12. He sees there are no event symbols shown for FIC-102 for the period...so he's assured that there were no alarms, failures, or downloads of the module that might be related. He decides to repeat the "Back...Copy trend pop-up...Forward...Paste to history control" sequence a couple more times to place the three odd trends he'd found on the trend pop-ups on the same chart.



13. "The boss isn't going to believe that the FIC-108 flow peaks like that. I told him the blending controls go goofy when we have disturbances like these." He knows what the process engineer is going to ask for when they meet to discuss this problem...so he clicks on the minimum value button for FIC-102, which sets a chart cursor, and clicks on the maximum value button for FIC-108, setting another chart cursor. With the two cursors set, the chart now shows the clock time for the two cursors and the



14. "That's what I'm talkin' about!" He hits the Copy button on the trend control, then clicks on the button to open the Operations Journal, creates a new entry, and hits the paste button. An image of the trend control appears in the Journal entry, and the three module names in the the trend control image are automatically added to the "Keys" list for this journal entry (so other folks on site will be able to find this entry looking for information related to any of these modules). He chooses a entry type and priority for this journal entry, and chooses the appropriate "Route To" lists for this type of thing...adds a few brief comments of his own and completes the entry.
15. "Probably worth keeping an eye on this for the rest of my shift" he thinks...so he hits Last Display to get to the process graphic, opens two faceplates (floating panel displays) for the two flow loops he's most likely to need to tweak, hits forward display to get back to the trend template. He then copies and pastes the two faceplates into two of the empty subpanels on top of the trend template, and hits the "Save Panel View" button. A dialog comes up to prompt him for a name for the (session) trend display he just created, suggesting "HalfTrend_Wed10:58". He overtypes part of the

name making it "FIC102n108Trend_Wed10:58" and dismisses the dialog. *(Later in the shift, he uses the Open Display browser, selects the session displays, and picks this trend display he'd saved earlier.)*

1.7.2 Displays and Workspace applications

While workspace displays and applications produce their displayable content through different means, they share space in the workspace framework. Usability is improved if they share certain features and behaviors.

1.7.2.1 Representing Information in the Same Way

It is desirable that the representation of certain kinds of information can be made consistent between the "built in" workspace applications and user configurable workspace displays.

1.7.2.1.1 User Needs

1. It should be possible to configure a system-wide mapping of alarm priority values (currently 3-15) to "background colors" and suitable "foreground colors".
 - Workspace applications that use color encoding to indicate alarm priority should use this configuration.
 - It should be easy to configure the color selection properties of graphic elements in user configured displays to use this configuration.
 - Pre-engineered displays that use color encoding to indicate alarm priority should use this configuration.

The implication is that if users change the system-wide alarm color mappings, that when that configuration is distributed, subsequent uses of workspace applications and displays that use the color mapping automatically follow the changes.

2. It should be possible to configure system-wide preferences for indicating unavailable data, when displaying data that is normally reflecting values that update in real-time. Configurable preferences should include:
 - An optional "unavailable data" background and foreground color override and opacity.
 - An optional "unavailable data" string substitution character.

It should be easy to configure "data link" type graphic elements in user configured displays to use these system wide preferences if desired. Pre-engineered displays will utilize this configuration to appropriately indicate unavailable data. Workspace applications that show values that update in real-time should use this configuration as appropriate.

3. Certain parameters related to DeltaV I/O and control functions have a "status" (ST) field, used to indicate data quality and certain control conditions. It should be possible to configure a system-wide preferences for indicating values with abnormal ST fields. Configurable preferences should include:
 - An background and foreground color overrides for ST field bit values.
 - An optional suffix string appended to the CV field strings when there is an abnormal ST field.
 - Precedence of ST field bit values when more than one abnormal condition is present.

It should be easy to configure "data link" type graphic elements in user configured displays to use these system wide preferences if desired. Pre-engineered displays will

utilize this configuration to appropriately indicate abnormal ST fields. Workspace applications that show data with ST fields should use this configuration as appropriate.

1.7.2.2 Runtime Workspace Health and Connection Status

The runtime workspace often serves as the connection operators have with the DeltaV control system(s) under their responsibility. They have an expectation that abnormal process conditions (alarms) will be indicated within very few seconds after they are detected in the system, so that problem analysis and (hopefully) resolution can begin very quickly.

While the runtime workspace is the top of many layers of services and computer servers, it is important that operators be able use it to quickly determine if the workspace is working normally, the hosting workstation service layers are operational, and that communications with the ACN is intact.

1.7.2.2.1 User Needs

1. A "workspace fault" data source should be provided. Any severe faults detected in this instance of the runtime workspace, or instance-specific service layers, would cause this data source to have a "bad" value²⁰. The intent is that when this indicates "bad", the operator is encouraged to perform a "workspace hard reset" operation (when the opportunity comes), in an attempt to clear the problem. If this does not clear the problem, the recommended action would be for the operator to move to a backup terminal or workstation.

Workspace hard reset operations clear this indicator; at least until another workspace instance fault is detected. No operator acknowledgment is necessary.

It will be possible for users to configure displays with data links to the "workspace fault" data source. "Workspace fault" will be indicated on the appropriate pre-engineered displays (e.g. the standard alarm banner display).

2. "A workstation/server fault" data source will be provided. Any severe faults in the primary services supporting this workspace instance would cause this data source to have a "bad" value. The intent is that when this indicates "bad", the operator should seek to have the primary services restarted or the workstation rebooted in an attempt to clear the problem. If this does not clear the problem, the recommended action would be for the operator to move to a backup workstation, or to a terminal served by another workstation/server.

Restarting primary services clear this indicator; at least until another fault is detected. No operator acknowledgment is necessary.

It will be possible for users to configure displays with data links to the "workstation/server fault" data source. It should also be possible to indicate the workstation name for which the fault is being indicated. "Workstation/server fault" will be indicated on the appropriate pre-engineered displays (e.g. the standard alarm banner display).

²⁰ Intermediate level(s) between "good" and "bad" may be appropriate, depending on the final implementation.

3. An "ACN connection fault" data source will be provided. Loss of regular communications ability²¹ with any (other) ACN node in the local zone would cause this data source to have a "bad" value. The intent is that when this indicates "bad", the operator should seek assistance in resolving network communications problems with the servers connecting him to the ACN, and to a backup workstation using an alternate connection route.

This indicator will clear when a viable communications path has been reestablished. No operator acknowledgment is necessary.

It will be possible for users to configure displays with data links to the "ACN connection fault" data source. It will be indicated on the appropriate pre-engineered displays (e.g. the standard alarm banner display).

4. A seconds-resolution, current time data source will be provided. The time will reflect the "operating system" time on the workstation providing primary services for this runtime workspace. The intent is to provide a convenient (predictably updating) "heartbeat" graphic element that operators can use to gain confidence that the services supporting the other real-time updating values are still "working".

It will be possible for users to configure displays with data links to the "current time" data source; while providing choices for formatting and local vs. UTC time. It will be indicated on the appropriate pre-engineered displays (e.g. the standard "tool bar" display).

1.7.2.3 Rearranging Displays and Applications

Runtime workspace features for rearranging displays and workspace applications let operators control the overall workspace content to match the task at hand. Objectives are:

- Predictable and repeatable behavior (important when operators are stressed)
- Minimize the effort involved in repositioning content
- Eliminate the need to manage (find, show, close, etc.) hidden content (windows)

1.7.2.3.1 User Needs

Panel Content

1. It will be possible to "close" the contents of a panel. Closing the content in a floating panel causes the floating panel window to disappear. Closing the content of a fixed panel causes the fixed panel to be displayed in its configured background color.
2. It will be possible when configuring a fixed panel, to designate it that its content cannot be closed. The content of such a fixed panel can be replaced by other content.
3. It will be possible to configure a button in user configured displays to close the current panel²².
4. All workspace applications provide a means to close themselves (i.e. their current panel).
5. If a panel is configured to have an "information and tool button" bar, one of the standard tool buttons that users may put in the bar is a "close this panel" button.

²¹ Including through any intermediate server "hops" in the communications topology.

²² A "close this panel" button would be a good candidate for the pre-engineered graphic element library.

6. To better support user configured toolbar displays, it will also be possible to configure a button in user configured displays to close another fixed or floating panel, identified by panel name. If the panel name can be resolved, the panel will close it's display or workspace application content.
7. Each panel can have at most one "open" content. When new content is opened (or pasted) to a panel, the previous panel content is automatically deactivated.
8. It will be possible to "copy" the content of a fixed or floating panel (display and most workspace application) and "paste" that content another panel. State information (layers visible, view center, scale, tree expansions) is also copied so that the copy renders as much as possible like the source panel.
9. It is possible to configure a "copy this panel" button in user configured displays.
10. Workspace applications which support copying to another panel provide a "copy this panel" button.
11. If a panel is configured to have an "information and tool button" bar, one of the standard tool buttons that users may put in the bar is a "copy this panel" button.
12. To better support user configured toolbar displays, it will also be possible to configure a button in user configured displays to copy another fixed or floating panel, identified by panel name.
13. It is possible to configure a "paste to this panel" button in user configured displays.
14. If a panel is configured to have an "information and tool button" bar, one of the standard tool buttons that users may put in the bar is a " paste to this panel" button.
15. To better support user configured toolbar displays, it will also be possible to configure a button in user configured displays to paste to another panel, identified by panel name.
16. The use of a "paste to this panel" button "consumes" the copied panel information. Subsequent "paste to this panel" actions have no effect on the destination panel, until after another "copy this panel" action is performed.

Floating Panels

17. When new content is opened in a floating panel, the the previous content of the floating panel is deactivated. The number of floating panels configured in a workspace framework determines the maximum number of floating displays that can be open at one time.
18. All floating panels appear with a control bar docked to the top edge of the user configurable portion of the floating display. The control bar always has a "close this floating panel" button.
19. Floating panels initially appear positioned at their anchor point (i.e. the top left corner of their control bar is located at the anchor point position.) Ideally anchor points are configured so that for the floating displays designed to appear there, no repositioning is necessary (not on monitor boundaries, clipped off, etc.) However, the operator will be allowed to reposition floating displays; anywhere in the bounds of the workspace framework when the framework is in "dedicated and controlled" mode.

20. Floating panels may be repositioned by mouse dragging. The drag source is the control bar (anywhere not occupied by a button).
21. Since free repositioning of floating panels could lead to some undesirable consequences (obscuring panel contents that should always be visible, "lost" floating displays, etc.), it will be possible to configure workspace framework setting where floating display repositioning is disabled. It will be possible to override this configuration setting in the launch information for the runtime workspace.
22. It will be possible to configure individual floating panels so that they offer no means of moving and/or resizing.
23. If there are multiple floating panels configured with a "category" that matches the content in a floating panel, the content may be reassigned to one of the other matching floating panels.
24. Up to four (left, right, up, and down) "reassign to next floating panel" buttons may appear (space permitting) in the control bar, depending on the number of eligible floating panels for this display. The button faces are chosen to indicate the relative direction of the anchor point for the floating panel they would reassign the display to.
25. When a "reassign to next floating panel" button is selected, the floating panel content (along with its state information) is moved to the closest eligible floating panel in that "direction". If the destination floating panel was already open, its old content is deactivated.

Sub-panels in Displays

26. Users may configure specialized "sub-panel" displays. Panels in sub-panel displays have their own information and tool button bars, that include at least the standard buttons for:
 - "close panel content"
 - "copy panel contents"
 - "paste to panel"
27. Sub-panels in sub-panel displays support the same close contents, copy contents, and paste behaviors as described for framework fixed panels.

1.7.2.4 Pop-up Dialogs and Menus

In most cases, display area is too precious to waste on user interface elements for infrequently used actions. Pull down and cascading menus are an effective backup to tool button bars when button space runs short. For more complex interactions, specialized pop-up dialogs, launched "in context", provide an space-optimized user interface.

1.7.2.4.1 User Needs

Pop-up Dialogs

1. To conserve display area, certain user interface actions are best performed by "pop-up" dialogs. (e.g. changing mode in a module function block, adding a display alert, etc.) The runtime workspace provides a number of "hard-wired" pop-up dialogs to accomplish certain actions, and configurable pop-up dialogs may be created²³.
2. Pop-up dialogs float over workspace fixed panels and compete with floating panels in the z-order.
3. Pop-up dialogs have no default position configuration, nor are they matched up with anchor points. Ideally they should initially appear next to (to the right of) the button

²³ No doubt that some of the pre-engineered displays provided will also use some pre-engineered (though re-configurable) pop-up dialogs.

- or other graphic element that caused them to open, in order to reinforce the "context" of which they are a part. Their positioning is constrained so that they never go off the edge of the workspace.
4. In most cases, pop-up dialogs should not be modal (thus allowing interactions with other parts of the workspace while they are on display). It is possible for user configured pop-up dialogs to be designated as modal, but generally this should be avoided except for delivering critical messages to the operator that require immediate attention.
 5. Pop-up dialogs have a control bar (one is attached to the top of user configurable ones) which provides a mechanism to close the dialog. It may often be appropriate that closing the dialog also occurs by other dialog interactions (e.g. "OK" button performs some action AND closes the dialog).
 6. Pop-up dialogs that lose their "parent context" (i.e. the graphic object that launched them (fixed panel display, floating panel display, parent pop-up dialog) is closed) are automatically closed.
 7. Since most pop-up dialogs are not modal, the number of open pop-up dialogs could get out of control. Therefore, any "parent context" for a pop-up dialog (fixed panel display, floating panel display, or a parent pop-up dialog) is constrained to have at most one immediate child pop-up dialog open at a time. If a parent context attempts to open another pop-up dialog while another is already open, the earlier pop-up dialog is first closed before opening the new one.
 8. Pop-up dialogs may be repositioned while they are open. They may be repositioned by mouse dragging; the drag source is the control bar (anywhere not occupied by a button).
 9. However, non-modal pop-up dialogs (especially ones that don't display the contextual information) that are repositioned far from their parent context could create a confusing situation for the operator. By default, repositioning is limited so that some part of the pop-up dialog may (slightly) overlap the parent context that opened it. Optional positioning hints provided by the graphic element (display logic) that opens a pop-up dialog can override the default behavior, by specifying either "no repositioning" or "unlimited repositioning".
 10. Repositioning a pop-up dialog with subordinate pop-up dialogs is allowed (subject to the repositioning constraints). When this occurs, the dialog and all of its subordinates are repositioned by the same amount. Again, repositioning of all pop-up dialogs is constrained so that they never go off the edge of the workspace.
 11. Pop-up dialogs associated with fixed or floating panel displays which are copied and pasted, are NOT also copied.

User Configurable Menus

We can anticipate a number of situations where users (and we) would like to provide more command options than there is room for buttons. Also, beyond a certain number of buttons, the button icons become indistinguishable, and most users need "tool tips" text to assure them they are about to activate the intended thing.

We can expect to face this problem in workspace applications, and a menu system is an effective and familiar solution. User configurable menus are tools to solve this problem in user-configurable (and pre-engineered) displays and workspace framework operations.

12. It will be possible to configure menu objects for use in the runtime workspace. These are named objects (like display definitions) which can be used in one or more contexts in the runtime workspace. We expect them to be used for:
 - Cramming more action "buttons" in space limited area (including information and tool button bars)
 - High density display launching lists
13. Menu objects consist of a collection of top level (peer) menu items. Each menu item has:
 - A menu item description string (used to select the menu item)
 - (Optional) icon bitmapEither:
 - a list of subordinate menu items (and menu list spacers)or
 - a menu item availability expression (default is "TRUE")
 - a menu action
14. It will be possible to configure buttons in user configured displays to "open" a menu object, by name. The menu object is displayed adjacent (by default immediately below) to the button that launched it.
15. It will be possible to configure buttons that appear in framework panel information and tool button bars and floating panel control bars to "open" a menu object, by name. The menu object is displayed adjacent (by default immediately below) to the button that launched it.
16. Menu object operation is similar to window menus. Depending on the configuration:
 - The top level menu item list appears horizontally.
 - Subordinate menu item lists appear vertically, with the optional icon
 - There is an indication if a subordinate menu item is an "end action" or has further subordinates
 - When an "end action" menu item is selected, its action is performed and all levels of the open menu object are closed.
17. Menu objects float over all workspace panels and open pop-up displays. They are essentially non-modal, but clicking anywhere in the workspace other than on a menu item closes all levels of the menu. Thus, one active menu object can be open in the workspace at any time.
18. Configurable menu actions include:
 - Launching displays, providing the most typical lunch context information
 - Launching a pop-up dialog, providing the most typical lunch context information
 - Perform typical display scaling operations on a specified panel, or "this" panel
 - Setting a display logic variable to a constant value
 - Running a display logic script

1.7.2.5 "Last Display" Features

A "go back to the last display" facility has traditionally proven valuable for operators. It provides an easy way to "recover" from an undesired display choice (e.g. mistake typing a display name or in choosing an unintended display link).

When combined with a "go 'forward' to the display I 'last display'd' from", it provides an ad hoc means for an operator to assemble a small number of (potentially unrelated) displays into a (temporary) surveillance group.

1.7.2.5.1 User Needs

1. The workspace will maintain a "last/forward display" stack for each framework fixed panel and each sub-panel in active sub-panel displays. (Floating panels do not have "last/forward display" stacks.)
2. The display stack records enough information (display definition, launch information, scaling, scroll position, layers being displayed, tree expansions, etc.) about the workspace displays or workspace applications²⁴ when they are replaced by newer content, so that the previous display/application can be re-opened in a visual state that closely resembles its state when it was replaced.
3. The "last/forward display" stack for each panel will hold information for at least 10 workspace displays or application instances.
4. It will be possible to configure "Last Display" and/or "Forward Display" buttons in user configured displays.
5. If a panel is configured to have an "information and tool button" bar, users may put "Last Display" and/or "Forward Display" button in that bar.
6. To better support user configured toolbar displays, it will also be possible to configure a button in user configured displays to perform a "Last Display" for another panel, identified by panel name.
7. Since the "last/forward display" stack is not infinite in size, there are situations where performing "Last Display" or "Forward Display" operations will have no effect. Each panel provides data sources for "can perform Last Display" and "can perform Forward Display". Users may configure links in user configured displays to these data sources to alter the appearance of their "Last Display" and "Forward Display" buttons (e.g. hidden or subdued if they can't be used). The data links may be addressable for "this panel", and for a specific panel identified by name.
8. Each time new panel contents are opened, the current content/state of the panel is captured as a "Last Display" in the "last/forward display" stack, before opening the new content.
9. When a "Last Display" operation can be performed in a panel (there are "Last Displays" in the "last/forward display" stack), the operation captures the current content/state of the panel as a "Forward Display" in the stack, closes the current panel contents, and uses the most recent "Last Display" information from the stack to open that display or workspace application.
10. When a "Forward Display" operation can be performed in a panel (there are "Forward Displays" in the "last/forward display" stack), the operation captures the current

²⁴ Sufficient "last display" state information for workspace applications will probably be quite different for each application, and different than that for workspace displays.

content/state of the panel as a "Last Display" in the stack, closes the current panel contents, and uses the most recent "Forward Display" information from the stack to open that display or workspace application.

11. The individual "last/forward display" stacks for each sub-panel in a sub-panel display are not preserved when the sup-panel display is replaced.
12. Saving a sub-panel display as a named instance does not save the individual "last/forward display" stacks for each sub-panel.
13. The "last/forward display" stack is preserved with changes of DeltaV user log-on (fast user switching). The "last/forward display" stack is cleared if a "workspace hard reset" is performed or the workspace application is terminated.

1.7.2.6 User Preferences and Presentation Persistence

Providing operators more control over the combination and presentation of content displayed in the runtime workspace, raises the issue: "should display content/presentation settings be preserved for some level of reuse"?

1.7.2.6.1 User Needs

1. Unless being recalled via the "last/forward display" stack, panel displays appear with their "as configured" presentation settings (scale, view center, layers displayed, etc.).
2. It will be possible to configure a "Save Panel View" button in user configured displays.
3. If a panel is configured to have an "information and tool button" bar, users may put a "Save Panel View" button in that bar.
4. To better support user configured toolbar displays, it will also be possible to configure a button in user configured displays to perform a "Save Panel View" for another panel, identified by panel name.
5. When a "Save Panel View" action is performed, a standard workspace dialog appears (centered on the affected panel), providing a means for the user to enter a session-scope display name for the current content and view settings of the panel. A suggested display name, derived from the current panel content, with a suffix for uniqueness, should be offered.
6. If the "Save Panel View" dialog is completed, the new display is created in temporary user-session display storage. For the remainder of that DeltaV user session (until the current DeltaV user logs off) the new display may be opened by the assigned name. When opened, the display initially appears with the same presentation settings (scale, view center, layers displayed, etc.) as it had when the "Save Panel View" action was performed.
7. When a "Save Panel View" action is performed for a panel containing a sub-panel display, the current content and presentation settings for all sub-panels are included in the new display saved.
8. Unless a "Save Panel View" action is performed to a new temporary user-session display, the presentation settings for a panel are lost once that display is pushed out of the "last/forward display" stack.

1.7.3 Using Displays

1.7.3.1 Resolving Display Definitions

Workspace display definitions hold the result of the display configuration process: a named collection of configurable properties and display components (with layout and presentation properties) along with associated display logic; sufficient to create an instance of a framework display, or pop-up dialog.

For operational consistency, the same display definitions will usually be shared by (distributed to) many or all of the runtime workspaces running in a DeltaV system. To save display engineering effort, many display definitions will be designed to run adequately (and thus should be shared) by workspaces running with different sizes and types of display hardware.

However, there are several situations where all workspaces sharing the same display definitions may not be desirable:

- A display definition which has been changed should probably receive "live system testing" on a single workstation before being "released to production" and distributed to all workstations. However, the display that has been changed has many linkages (which should be tested) to other displays which have not changed (and are released for general use.)
- A user may feel more comfortable using the English display definitions on his workstation, even though the primary language chosen for the DeltaV system when installed was non-English.
- Some workstations in the system are being upgraded to new 16:9 monitors, and some of the most frequently used displays have versions optimized for workspace framework designed for this display hardware.
- While troubleshooting a customer complaint, a technical support engineer wants to run the workspace using the pre-engineered displays that came with the DeltaV software²⁵, rather than the ones the user has modified.

These needs suggest a solution where:

- Its possible to have distinct display definitions with the "same" name used to open them.
- Most instances of running workspaces in a DeltaV system want to use the "general use" display definitions.
- Specific workspaces (doing display testing, running on unusual display hardware, being used by an atypical user) need a means to override the normal display definition resolution process with one specific to the purpose at hand.

1.7.3.1.1 User Needs

1. It should be possible to configure a standard display definition resolution "search path" for each workspace framework. The search path establishes the sequence of locations searched to resolve a display definition name. For example, for a search path like:

Released

²⁵ Suggests a need for a means to segregate the pre-engineered displays that came with DeltaV from the ones that users have built (even to replace the pre-engineered displays).

DvStd\SysLang\43Mon

and assuming the workspace had been started with sufficient information to tell it that the root storage for all display definitions it should use was "C:\DeltaV\DVDATA\DDF", attempts to open a display with display definition name "AlarmBanner.ddf", would look for it first in:

c:\DeltaV\DVDATA\DDF\Released\AlarmBanner.ddf

and if not found there, would look for it in:

c:\DeltaV\DVDATA\DDF\DvStd\SysLang\43Mon\AlarmBanner.ddf

In this case the pre-engineered displays that came with DeltaV would be used except for those that the user chose to override by putting them in the directory listed earlier in the search path.

2. It should be possible to alter the display definition search path for a workspace framework by providing additional information when the framework is launched. For example, assume the search path configuration for a framework:

```
Released/%Language%/%DisplayHW%/%Skin%
Released/%Language%/%DisplayHW%/Default
Released/%Language%/43Mon/%Skin%
Released/%Language%/43Mon/Default
Released/SysLang/%DisplayHW%/%Skin%
Released/SysLang/%DisplayHW%/Default
Released/SysLang/43Mon/%Skin%
Released/SysLang/43Mon/Default
DeltaVStd/%Language%/%DisplayHW%/Default
DeltaVStd/%Language%/43Mon/Default
DeltaVStd/SysLang/%DisplayHW%/Default
DeltaVStd/SysLang/43Mon/Default
```

and the workspace running on a 16:9 aspect ratio monitor, had been started with launch information including:

DDF_PATH="%DisplayHW%='169Mon' "

After substitutions and eliminating paths with unresolved symbols, the effective search path for this workspace becomes:

```
Released/SysLang/169Mon/Default
Released/SysLang/43Mon/Default
DeltaVStd/SysLang/169Mon/Default
DeltaVStd/SysLang/43Mon/Default
```

This essentially tells to workspace to look for display definitions in the /169Mon/Default directories first, and if displays optimized for the 16:9 monitors can't be found, use the old 4:3 aspect ratio displays.

3. It should be possible to interactively view and alter the display definition search path. For example, with the search path configuration of the last example, a tech support engineer (more comfortable using English) starts an instance of the workspace with launch information including:

`DDF_PATH="%Language%='EN' "`

After substitutions and eliminating paths with unresolved symbols, the effective search path for this workspace becomes:

`Released/EN/43Mon/Default`
`Released/SysLang/43Mon/Default`
`DeltaVStd/EN/43Mon/Default`
`DeltaVStd/SysLang/43Mon/Default`

He is getting the English displays when available, but he notices unexpected behavior in the alarm area filter display, and sees that the user has modified the standard one that came with DeltaV. He pushes the "Advanced Features" button in the TOOLBAR panel, which opens a user menu which gets him to a "Display Search List" menu item. He pushes it and a standard workspace dialog appears showing the current display resolution search list.

He un-checks the list item for `DeltaVStd/SysLang/43Mon/Default` (since this was installed as a German system, he's not interested in the standard DeltaV German language displays). He then selects the list item for `DeltaVStd/EN/43Mon/Default`, and moves it to the head of the list. Now he can test with the standard (English) DeltaV alarm displays on this workstation, until he gets this problem resolved.

The "Display Search List" standard dialog supports:

- Reordering the current search list
 - Adding new items to the search list (probably with browsing)
 - Disabling items in the search list (allowing them to be enabled again easily)
4. Alterations made via the "Display Search List" dialog are maintained until that workspace instance closes, or a "workspace hard reset" is performed. After a workspace hard reset, the display definition search list is re-resolved per the current workspace framework configuration available, and the original launch information.
 5. It is expected that the display engineering application(s), will steer the user toward saving moves "under development" to a well known "Test" directory on the local workstation (a directory whose contents do not participate in the automatic display configuration distribution system.)

When the display engineering applications go into display test mode, they run like a normal workspace, but the Test directory is added to the head of the display definition search list. Thus the displays in the Test directory can be tested, (with each other, and with displays already distributed in the "Released" directories) on that workstation, without impacting display behavior on the rest of the system.

When the display engineer is satisfied that the display changes are sound, he (essentially) moves the displays from the Test directory to the appropriate Released directory, and (per the display distribution preferences configured) they are made available to all appropriate workstations in that DeltaV system.

1.7.3.2 Opening New Displays

1.7.3.2.1 Locations for New Displays

By default, new display instances appear in the workspace at a location determined by matching the display definition (configuration) to the workspace configuration (and current content). However, a means to allow display logic to override the default workspace behavior is required.

1.7.3.2.1.1 User Needs

1. It will be possible to configure graphic elements (e.g. buttons) on displays and menu items to open new instances of displays in the workspace.
2. It will be possible to open a display in a specified workspace panel (fixed or floating).
3. It will be possible to override the default display location in the following ways:
 - Direct a display to "this panel" in the framework.
 - a. If from a fixed or floating panel display, "this panel" is the panel the "from" display is already on. (The "from" display is replaced).
 - b. If from a menu, "this panel" is the panel from where this menu (or one of it's predecessor displays or menus) was opened.
 - Direct a display to a specific framework panel or anchor point by name.

1.7.3.2.2 New Display Scaling

We expect users will create workspace frameworks that "nicely fit" their most frequently used display devices, and in turn will generally create display definitions that "nicely fit" their workspace framework (panel sizes and anchor point positions). The as-configured display definition size will be considered the "native" size for a display. We'd expect the user to choose a native size for a display that would require minor scaling adjustments for it's primary intended use.

Even with careful display engineering, minor scaling adjustments may often be required so that displays exactly fit the panels they should appear in (adjusting for minor changes in panel border widths, presence of information and tool button bars, etc.) For the most pleasing presentation (and avoidance of nuisance scroll bars) the workspace should automatically scale displays to exactly fit their destination panel, but not perform "extreme" scaling to fit.

In addition, supporting simple overall scale adjustments to the "native size" of displays rendered could provide a means for a user to work effectively with a unusual screen resolutions, or with visual acuity issues.

1.7.3.2.2.1 User Needs

Display Layer Visibility and Scaling

1. It will be possible to configure layers in a display so they are not shown when the display is rendered below a specified scale factor (a value in the range of 10% to 1000%) relative to the native size.

Overall Scale Factor

2. It will be possible to apply an overall scale factor to displays that appear in the workspace. The overall scale factor (with separate factors for horizontal and vertical scale adjustments) is applied to the native size of displays (panel, and pop-up) as the first step in determining the final scaling of the new display to be rendered. Overall scale factors from 10% to 1000% will be supported.
3. The overall scale (horizontal and vertical) factors is a configurable property for each workspace framework (defaults to 100% horz and 100% vert). The overall scale factors for a workspace can be overridden by launch information when the workspace application is started.
4. It will be possible to change the overall scale factors for a workspace after it has started. It will be possible to configure a button in user defined displays or menu items to open the workspace "change overall scale factor" dialog. When a workspace hard reset is performed, the overall scale factors revert to those defined at workspace startup (launch info if present, otherwise as configured.)
5. The "change overall scale factor" workspace dialog provide a means to see and alter the horizontal and vertical scale factors. When the changes are accepted, all currently open panel displays re-render themselves using the new overall scale factors in their scaling operations.

Scale to Fit Panel

6. After the overall scale factor is applied, additional scaling is automatically computed to cause displays to exactly fit the destination framework panel (ideally no scroll bars, and no gaps horizontally or vertically.)
7. Extreme scaling is probably objectionable, so the workspace will apply "scale to fit panel" constraints:
 - Shrink limit: range: 10% to 100% (applies to both horizontal and vertical)
 - Grow limit: range: 100% to 1000% (applies to both horizontal and vertical)
 - Distortion limit: range: 0% to +/- 100% (acceptable change in the aspect ratio)

If "scale to fit panel" scaling cannot be accomplished within the constraints, horizontal and vertical scaling within the constraints will be chosen, resulting with a display not filling the panel in either/both dimensions and/or scroll bars.

8. The "scale to fit panel" constraints are configurable properties for each workspace framework. They can also be overridden by launch information when the workspace application is started.
9. It will be possible to change the "scale to fit panel" constraints for a workspace after it has started. It will be possible to configure a button in user defined displays or menu items to open the workspace "change scale to fit panel constraints" dialog. When a workspace hard reset is performed, the constraints revert to those defined at workspace startup (launch info if present, otherwise as configured.)
10. The " change scale to fit panel constraints " workspace dialog provide a means to see and alter the constraints. When the changes are accepted, all currently open panel displays re-render themselves using the new constraints in their scaling operations.

Last/Forward Displays Scaling

11. Displays re-opened via the last/forward display stack for a panel do not have "overall" or "scale to fit panel" scaling applied. The actual scale (horizontal and vertical, including the result from any manual scaling operations that had been performed)

when the display was last deactivated is stored in the last/forward display stack, so it can be again applied when the display is re-opened.

12. The actual scaling information stored in the last/forward display stack is not affected by the "change overall scale factor" workspace dialog, or the "change scale to fit panel constraints" dialog.

1.7.3.2.3 Opening Displays from other Displays

One of the easiest (and most frequently used) means to open a new display instance is via another display already appearing in the workspace. Users will often use graphic elements (i.e. buttons) on displays specifically to provide easy access to related displays, or specialized display "directories".

1.7.3.2.3.1 User Needs

1. I will be possible to use a button to open another display instance. Optional button configuration will:
 - Allow overriding the default location for the new display instance.
 - Resolve the name for certain "well known" displays (i.e. primary control, faceplate, detail) from a module name.
 - Provide a means for additional launch information so that tag resolution can be performed.
2. I will be possible to click on a grouping of graphic elements (not necessarily a button) in a display and have the click action open another display instance. Optional configuration will:
 - Resolve the name for certain "well known" displays (i.e. primary control, faceplate, detail) from a module name.
 - Provide a means for additional launch information so that tag resolution can be performed.

1.7.3.2.4 By Name

Its quite possible that certain displays may not be easily accessible from other displays. A means to browse and select from all available displays (definitions) is necessary.

1.7.3.2.4.1 User Needs

1. It will be possible to configure a button in user configurable displays to open the standard display selection dialog.
2. In systems connected to other DeltaV zones, the display selection dialog will allow the selection of another zone²⁷; defining the scope for display definition browsing and selection. The default zone selection is "this zone". If no other zones are available, zone selection does not appear.
3. The display selection dialog provides the following types of display selection:
 - **Standard displays:** Display definitions in the selected zone that require no further "tag resolution".

²⁷ Constrained to zones which have been configured to expose their displays for use by this zone.

- **Tag displays:** Display definitions in the selected zone that require one or more tags to be defined, so that a display instance can be opened²⁸.
 - **Session displays:** Display definitions that have been created by (and private to) this instance of the runtime workspace application.
 - **Recent selection displays:** The last 20 **standard** or **tag displays** that have been selected by this dialog. (There is one "recent selection" history maintained in the runtime workspace²⁹.)
4. When selecting **standard** or **tag displays**, the display definitions are shown using the current display resolution search path. Display definition names from all search paths are combined. By default, display definitions not accessible due to search order are not shown. It will be possible to override the default, and have all display definitions shown (some distinguishable mainly by their "search path"), thus providing a means to override the normal search path in this instance..

If displays are organized by sub-folders, the sub-folder names also appear; allowing the user to drill-down into the display definitions stored in subfolder(s) with that name.

Standard or **tag display** definition details may be shown (and sorted on) including:

- display definition name
 - last modification timestamp
 - search path
5. When selecting **tag displays**, it is possible to identify a specific tag display definition, and see the tag(s) that may be resolved. The tag(s) are identified by user-configurable comments (e.g. "Module name: "; intended to help the operator enter appropriate strings for tags to be resolved.) (The tag substitution strings entered will be included in the launch information when the tag display is opened.)
6. When selecting **session displays**, details may be shown (and sorted on) including:
- session display definition name
 - saved timestamp
 - saved by username
7. When selecting **session displays**, a means to delete session display definitions (individual or multiple selection, or a "remove all" mechanism).
8. When selecting **recent selection displays**, details may be shown (and sorted on) including:
- display definition name
 - (if a tag display) tag string(s)
 - search path
 - last selection timestamp
9. When selecting a **tag display** from the **recent selection display** list, it is possible to see the tag strings that were last used. The same tag strings can be used again, or they may be replaced.

²⁸ Used for faceplates, detail displays etc. where one (or a small number of tags) can be used to dynamically create a display instance "resolved" for the specific "target".

²⁹ The "recent display selection" history has nothing to do with the per-panel last/forward display stacks.

1.7.3.2.5 From Controls and Workspace Applications

Workspace application and certain workspace "controls" (which may be placed in user configured displays; e.g. an alarm summary control) may request that new displays be opened in the workspace³⁰.

1.7.3.3 *Panning and Controlling Display Scaling*

Ideally, using displays designed for a workspace framework will eliminate the need for most manual display scaling or panning (using scroll bars) operations. Using displays in frameworks (devices) for which they weren't designed, will require some means to control scaling, and which parts of the display are currently visible.

1.7.3.3.1 User Needs

1. The following manual scaling operations will be accessible for displays in panels:
 - **View All:** all parts of the display are visible within the panel. Display occupies full panel area in one dimension and may have a gap in the other dimension (display distortion constraint still applies.)
 - **Fill Panel:** Display occupies full panel area in one dimension and may have a scroll bar in the other dimension (display distortion constraint still applies.)
 - **Reset:** Display is redrawn as if opened for the first time in the panel (Current "scale to fit panel" constraints are applied.)
 - **Nudge Larger:** current display scale (both horizontal and vertical) is increased by 10%. Will increase the scale until it reaches approximately 1000% of the native size. (Current aspect ratio is maintained)
 - **Nudge Smaller:** current display scale (both horizontal and vertical) is decreased by 9.1%. Will decrease the scale until it reaches approximately 10% of the native size. (Current aspect ratio is maintained)
 - **Native :** display rendered at it's native size and aspect ratio; no overall scale factors or "scale to fit panel" constraints are used.
 - **Set Display Scale dialog:** causes a standard workspace dialog to appear, letting the user see and set the horizontal and vertical scaling factors for this display (overriding any "scale to fit panel" constraints.)
 - **Region zoom-in:** an operation where the user indicates a rectangular region in a framework panel, and the display scaling is increased and the display scrolled to center the view on the indicate region. (See details below.)
2. It will be possible to create buttons in user configured displays for any of the panel scaling operations listed above. It will be possible to create buttons for any of the scaling operations in the information and tool button bars for workspace panels. It will be possible to perform any of the panel scaling options from a user configured menu item.

It will be possible to configure the scaling operation buttons (or menu items) to direct the operation to "this" panel, or to another panel identified by name.
3. Workspace panels show horizontal and/or vertical scroll bars, when there are parts of the display that are not currently visible in the panel. The scroll bars indicate the proportion of the display is currently visible in that dimension, as well as the current

³⁰ Since the workings of these actions are not (directly) user configurable (visible), the details are appropriate for design documents.

scroll position. Typical scrollbar clicking and dragging operations are used to move the visible portion of the display.

4. It will be possible to perform a "region zoom-in" operation, which combines a scale adjustment operation with panning, to show a close-up of an indicated area of a display in a framework panel.
5. A "region zoom-in" operation requires several steps:
 - The operation is initiated via a button or menu item. The mouse pointer changes to indicate a region selection action is pending.
 - The user clicks in the framework panel to establish one corner of the region. The mouse pointer changes to indicate that a region selection action has been started.
 - The user clicks in the same framework panel to establish the opposite corner of the desired region.(When appropriate pointing hardware (mouse or stylus) is present, the last two steps can be combined by a pointer "drag" operation.)
6. Once the desired rectangular region has been indicated, the display in that panel has its scale adjusted so the entire rectangle will just fit (centered in) the panel area. (Scale to panel constraints are not applied, the current aspect ratio is maintained.)

(Support special mouse operations for panning (e.g. drag panning mode)? Take advantage of 2 and 3 button pointing devices? Useful mouse wheel behaviors?)

1.7.3.4 Interactive Elements in Displays

1.7.3.4.1 "Perform Action" Buttons

Momentary "perform action" buttons are one of the simplest (and most obvious) interactive elements in user configured displays.

1.7.3.4.1.1 User Needs

Workspace Action Buttons

1. It will be possible to configure "perform action" buttons in user configurable panel, or popup displays to perform workspace functions (many identified in previous sections.) Workspace actions include:
 - Close display
 - Copy this display
 - Paste display to panel
 - Snapshot panel contents
 - Launch detached window (from panel)
 - Show last display
 - Show forward display
 - Scaling operations:
 - View all
 - Fill Panel
 - Reset
 - Nudge Larger
 - Nudge Smaller
 - Native size
 - Region zoom-in
 - Show "overall scale factor" dialog

- Show "scale to fit panel constraints" dialog
 - Show "display scale" dialog
 - Save panel view
 - Switch workspace mode (dedicated vs. windows application)
 - Workspace hard reset
 - Show "display selection" dialog
 - Show "display search list" dialog
 - Show "display layers" dialog
 - Show "add display alert" dialog
 - Add alarm profiles
2. Workspace "perform action" buttons are available in two standard sizes: small (match the appearance of workspace action buttons in information and tool bars for workspace panels), and large (larger versions of same). Standard size buttons are displayed with standard bitmaps corresponding to the workspace action they perform.
 3. It will be possible to configure custom size workspace "perform action" buttons; where the user choose the size/shape of the button rectangle. The user may choose a text legend to appear on these.
 4. By default, a standard (localized) tool tip string appears when the mouse pointer hovers over workspace "perform action" buttons. Users may change the string or disable the tool tip behavior for a button.

Custom Action Buttons

5. It will be possible to easily configure any of the following custom action buttons for use in displays:
 - **Open Display:** opens a new display instance for a specified display definition (panel, or pop-up); may optionally indicate a specified destination panel or anchor point.
 - **Open Primary Control Display:** opens a new display instance of the primary control display for a specified module; may optionally indicate a specified destination panel or anchor point.
 - **Open Faceplate Display:** opens a new display instance of the default faceplate display for a specified module; may optionally indicate a specified destination panel or anchor point.
 - **Open Detail Display:** opens a new display instance of the default "details" display for a specified module; may optionally indicate a specified destination panel or anchor point.
 - **Open Application:** opens the specified workspace application; may optionally direct the display to a specified destination panel or anchor point.
 - **Open Menu:** opens the specified user configured menu.
 - **Open Operations Journal:** opens the operations journal application, for the specified system object (module/node/device, etc.)
 - **Write value:** writes a configured (constant) string value to a workspace variable or to a real-time data source (i.e. DeltaV system); optional confirmation before write.
 - **Increase value:** writes a float value that is a specified x% of range larger than the current value read, constrained to the range; optional confirmation before write.

- **Decrease value:** writes a float value that is a specified x% of range smaller than the current value read, constrained to the range; optional confirmation before write.
6. It will be possible to configure the size of custom action buttons.
 7. It will be possible to configure a text label for custom action buttons. The text may be a constant string, or a reference to a workspace variable or real-time data source³¹. The foreground and background colors are configurable; either constant colors, or determined by color information provided by the containing graphic element³².
 8. A tool tip string can be configured to appear when the mouse pointer hovers over custom action buttons.

1.7.3.4.2 Selection Targets

Interacting with displays by pointing at graphical elements can provide a powerful and efficient user interface for operators. The desired responses might include:

- Identification: the process control "tag" associated with the graphical element might appear briefly (like a tool tip), or appear in the "information and tool button" bar for the panel.
- Setting "focus" so that workspace applications (or other workspace displays) designed to "track focus" can adjust their content automatically. (For example: a custom graphic schematic of DeltaV controllers arranged by their physical location in the plant; and the Diagnostics workspace application (in tracking mode) automatically showing the top level integrity information for each controller as it is selected.)
- Exposing more information or UI controls: for displays cramped for space, less frequently used data links or "perform action" buttons might remain invisible until the graphic element they are associated with is selected.
- Invisible buttons: the selection region for the graphic element can serve as an invisible perform action button used to navigate to another display or open a pop-up display with more data links and/or action buttons for the selected graphic element.

1.7.3.4.2.1 User Needs

1. It will be possible to configure "selection targets" in displays. The selection targets are defined as a rectangular region.
2. Other configurable properties:
 - **Selected data path:** a string which provides the "data link path" to the object (module, node, DST, FF device, or (field of) parameter therein) that this selection target selects. If omitted, the selection target has no actual selection behavior...it just behaves as an invisible button for the **selection action**.
 - **Selection action:** (optional) Selection targets may be configured to perform any of the workspace or custom action button functions listed previously.
 - **Tool tip string** (optional) When specified, appears when the mouse pointer hovers over the selection target.
 - **Two step select for action:** boolean property that controls the selection behavior:

³¹ Button surface bitmaps or more elaborate animations would be possible combining our basic custom action button with other graphic elements.

³² Expectation that color animation behavior would come from a "grouping" element.

- If TRUE: The action is performed if the selection target is selected again while already selected. (After the first selection, the selection target shows "is selected" focus, the workspace captures the "selected data path" in its variables, and the tool tip (if any is displayed).
 - If FALSE: The action is performed immediately upon (first) selection. Information provided when the workspace instance is launched can override this per-target configuration setting; forcing two step actions for all selection targets.
3. At runtime, selection targets have two states: selected and not-selected. When a new display instance is created, all selection targets are not-selected³³. The current selection state is available as a workspace variable (associated with this target in this display)³⁴.
 4. At most one selection target may be selected in a display instance. Selecting a new selection target (that has a "selected data path" configured) automatically de-selects any previously selected target in that display. (Each active display instance in the workspace may have its own selection target in the selected state.)
 5. When a selection target (that has a "selected data path" configured) is selected, the workspace provides a visualization that it has "selection focus". The properties for the "selection focus" visualization are configured for the workspace framework, and include:
 - Selection region border line properties (style, weight, color)
 - Selection region rectangle fill properties (transparency level, color)
 6. When a selection target (that has a "selected data path" configured) has a mouse pointer hovering over it, the workspace provides a "selection target present" visualization. The properties for the "selection target present" visualization are configured for the workspace framework, and include:
 - Selection region border line properties (style, weight, color)
 - Selection region rectangle fill properties (transparency level, color)
 7. If the regions for selection targets overlap, the selection target "on top" at the selection point, receives the selection (or the mouse pointer hover behavior); the target(s) below do not perform any of their selection behaviors.
 8. If the region for a selection target overlaps with other interactive graphical element (perform action buttons, data entry controls, etc.) the object "on top" where the pointer selection is made gets the selection. Ordinarily selection target regions would be configured "below" other interactive graphical elements...so that clicking in the area but avoiding the buttons and data entry fields would "hit" the selection target.
 9. Selection targets which are currently invisible (or inactive) cannot be selected, nor produce the "selection target present" visualization.
 10. A well known workspace variable (instances for each panel and anchor point) holds the "selected data path" string for the currently selected selection target. The

³³ Unless display launch information includes information requesting an automatic selection be performed.

³⁴ Expectation is that other graphical elements could have visibility behavior tied to the selection state of a selection target.

"information and tool button" bar for the panels can be configured to have labels linked to these workspace variables.

11. A well known workspace variable (instances for each panel and anchor point) holds the "tool tip" string for the currently selected selection target. Other workspace applications (or displays) tracking the selection focus in this display can access the tool tip string to help describe their current tracking state.

1.7.3.4.3 Two State Data Entry Controls

On/off type manipulations occur frequently in DeltaV control modules may also be used to interact with display logic for pop-up (dialog) displays.

1.7.3.4.3.1 User Needs

1. It will be possible to easily configure two-state data entry controls in displays. They provide a visual (one of two states) representation of the current value, and a means to change the current value.
2. Choices for visual presentation include:
 - Button ("pushed in" indicates the "on" state). Button size is configurable.
 - ☒ style checkbox (the "x" indicates the "on" state)
 - ☒ style checkbox (the "✓" indicates the "on" state)
 - ☐ style radio button (the "•" indicates the "on" state)
3. Configuration includes:
 - Reference to workspace variable or real-time data source that holds the value that the control represents
 - (optional) The value that represents the "on" state. If not specified, all values other than the "off" value are displayed as the "on" state.
 - (optional) The value that represents the "off" state. If not specified, all values other than the "on" value are displayed as the "off" state³⁵.
 - (optional) The value that should be written when the control is clicked (while in the "off" state) to turn it "on". If none supplied, it cannot be turned "on".
 - (optional) The value that should be written when the control is clicked (while in the "on" state) to turn it "off". If none supplied, it cannot be turned "off"³⁶.
 - If confirmation required before writing.
 - (optional) Tool tip string when in the "on" state.
 - (optional) Tool tip string when in the "off" state.Button style visual presentation configuration also includes:
 - Button label when in the "on" state.
 - Button label when in the "off" state.
4. Runtime response to clicking a two-state data entry control should be to give immediate feedback to the click action. (Changing the presentation between "on" and "off" may be delayed (or even refused) if the control is connected to a DeltaV parameter.)

³⁵ Either the "on" or the "off" value may be specified. If the current value cannot be matched to either "on" or "off" values when both are specified, the control becomes invisible and cannot be turned off.

³⁶ If neither "on" or "off" write values are configured, then the two state data entry control is "indicate only".

1.7.3.5 Numeric Data Entry Control

Numeric data entry is frequently required in interactions with regulatory control modules.

1.7.3.5.1.1 User Needs

1. It will be possible to easily configure numeric data entry controls in displays. They provide a visual representation of the current digital value, and a means to change it.
2. Choices for visual presentation include:
 - **"In place" data entry:** when selected, the control indicates selection focus and becomes a numeric data entry field (the current value is replaced by the empty data entry field and cursor). Keyboard numeric and Enter and Esc keys may be used to complete or cancel the data entry.
 - **Numeric data entry dialog:** when selected a standard workspace modal dialog appears providing numeric and Enter/Ctrl/Esc buttons³⁷. While the dialog appears the numeric data entry control continues to reflect the current value being read.
3. Configuration includes:
 - Reference to workspace variable or real-time data source that holds the value that the control represents.
 - (optional) Reference to workspace variable or real-time data source that holds the maximum value that may be entered. No maximum value constraint if not configured.
 - (optional) Reference to workspace variable or real-time data source that holds the minimum value that may be entered. No minimum value constraint if not configured.
 - If confirmation required before writing.And if the numeric data entry dialog form is chosen:
 - Dialog placement options

1.7.3.6 Alphanumeric Data Entry Control

Alphanumeric data entry is sometimes used to communicate general information or commands to control systems. Often, the acceptable entries are limited, so selecting from a list saves keyboard effort and avoids mistakes.

1.7.3.6.1.1 User Needs

1. It will be possible to easily configure alphanumeric data entry controls in displays. They provide a visual representation of the current string value, and a means to change it
2. Choices for visual presentation include:
 - **"In place" data entry:** when selected, the control indicates selection focus and becomes an alphanumeric data entry field (the current value is replaced by the empty data entry field and cursor).
 - **Alphanumeric data entry dialog:** when selected a standard workspace modal dialog appears. While the dialog appears the alphanumeric data entry control continues to reflect the current value being read.

³⁷ We may consider offering a means to switch to "four function calculator" mode in the numeric entry display...saving users having to pull out their calculator and rekeying in the results.

3. Configuration includes:
 - Reference to workspace variable or real-time data source that holds the value that the control represents.
 - If confirmation required before writing.And if the alphanumeric data entry dialog form is chosen:
 - Dialog placement options
4. For either visual presentation, the control (or dialog) can optionally be configured to provide a selection list. When a selection list option is chosen, related configuration includes:
 - Reference to workspace variable or real-time data source that holds the values that should appear in the selection list. (A comma delimited string is expected.)
 - A "constrain entries to only selection list values" property. When false, entries other than those in the selection list (may be typed in) will be accepted.

1.7.3.7 Specialized Display Components

1.7.3.7.1 Pop Up Dialog Displays

1.7.3.7.2 Sub-Panel Displays

1.7.3.7.3 Operator Prompt

1.7.3.7.4 Alarm Flood Management

1.7.3.7.5 Display Tree

1.8 Security Features

1.8.1 User Authentication

1.8.2 Fast User Switching

1.8.3 User Privilege Aggregation

1.8.4 Electronic Signatures

1.8.5 Inter-zone Credentials Wallet

Process Display Configuration and Distribution

1. PROCESS DISPLAY CONFIGURATION AND DISTRIBUTION	428
1.1 RELATED DOCUMENTS:	428
1.2 OBJECTIVES	428
1.3 KEY TERMS.....	429
2. REQUIREMENTS.....	430
2.1 SYSTEM TOPOLOGIES - SINGLE PC, DELTA V STYLE SYSTEMS, AND ZONES	430
2.1.1 <i>Single PC</i>	430
2.1.2 <i>Delta V Style Systems</i>	430
2.1.3 <i>Zones</i>	430
2.2 DISPLAY CONFIGURATION	430
2.2.1 <i>General</i>	430
2.2.2 <i>Look and Feel</i>	432
2.2.3 <i>Static Elements</i>	432
2.2.4 <i>Dynamic Elements</i>	433
2.2.5 <i>Display Storage</i>	433
2.3 BEST ENGINEERING PRACTICES	433
2.3.1 <i>Display Classes</i>	433
2.3.2 <i>Display References</i>	434
2.3.3 <i>Bulk Edit</i>	434
2.4 DISPLAY MANAGEMENT	434
2.4.1 <i>Explorer</i>	434
2.4.2 <i>VCAT</i>	435
2.4.3 <i>Download indicators</i>	435
2.5 DISPLAY DISTRIBUTION	436
2.5.1 <i>Display assignment</i>	436
2.5.2 <i>Display downloading</i>	436
2.6 RUNTIME STRUCTURE AND BEHAVIORS.....	436
2.6.1 <i>Runtime Structure</i>	436
2.6.2 <i>Runtime Behavior</i>	437
3. CONCEPT	438
3.1 OVERVIEW	438
3.2 DISPLAY CONFIGURATION	438
3.2.1 <i>Display Configuration Overview</i>	438
3.2.2 <i>Display Constructs</i>	439
3.3 BEST PRACTICES – DISPLAY CLASSES	440
3.3.1 <i>Background</i>	440
3.3.2 <i>Display Classes</i>	442
3.3.2.1 <i>Process Cell – Display Class “SALTS”</i>	442
NOTE : there are actually 3 levels of view of a Totalizer in the examples	442
• The one shown above called TOTALIZER-THUMBNAIL (no flow meter).....	442
• The one in REACTOR-DETAIL called TOTALIZER-OUTLINE	442
• The totalizer detail display TOTALIZER-DETAIL	442
3.3.2.2 <i>Unit – Display Class “REACTOR-DETAIL”</i>	443
3.3.2.3 <i>Display Class “TOTALIZER”</i>	444
3.3.2.4 <i>Display Class “REACTOR_OUTLET”</i>	445
3.3.2.5 <i>Display Class “LEVEL_METER”</i>	446
3.3.2.6 <i>Display Class “ON_OFF_VALVE”</i>	447

3.3.2.7	Display Class “FLOW_METER”	448
3.3.2.8	Summary of Display Class Relationships.....	449
3.3.3	<i>Display Instances</i>	450
3.3.3.1	Download Bindings	450
3.3.3.2	Instance Tree.....	451
3.3.4	<i>Display Class Downloads</i>	451
3.4	DISPLAY MANAGEMENT AND DISTRIBUTION.....	452
3.4.1	<i>Displays Folder and Sub-folders</i>	452
3.4.2	<i>Operator Subsystem</i>	452
3.4.3	<i>Download</i>	452
3.5	INTERFACE WITH THE RUNTIME	452
3.5.1	<i>Display Downloads</i>	452
3.5.1.1	Rich Client.....	453
3.5.1.1.1	dynamic.dvg.....	453
3.5.1.2	Thin Client.....	453
3.5.1.2.1	OpenDisplay.aspx & OpenDisplay.js	453
3.5.1.2.2	FunctionBlocks.js.....	455
3.5.1.2.3	dynamic.dvg.SVG/XMAL.....	455
3.5.1.2.4	dynamic.dvg.js.....	456
Figure 1. Display ‘dynamic’		439
Figure 2. Process Cell - Salts		440
Figure 3. Display Class – “SALTS”		442
Figure 4. Display Class – “REACTOR-DETAIL”		443
Figure 5. Display Class – “TOTALIZER”		444
Figure 6. Display Class – “REACTOR_OUTLET”		445
Figure 7. Display Class – “LEVEL_METER”		446
Figure 8. Display Class – “ON_OFF_VALVE”		447
Figure 9. Display Class – “FLOW_METER”		448
Figure 10. Display Class Relationships		449
Figure 11. Display Storage		453

Process Display Configuration and Distribution

This document describes Process Display Configuration and Distribution. It addresses the following topics:

- **Single PC, DeltaV Style Systems, and Zones** – The Process Display implementation needs to support Single PC through multiple Zone topologies.
- **Display Configuration** – look / feel, the configuration of static and dynamic graphic elements, and the integration with the overall configuration system (i.e. rename tracking, aliases, cross-referencing, etc).
- **Best engineering practices** – part of this will be achieved through the use of Display Classes. Associating display classes with module classes allows displays and binding tables to be defined just once on a Module Class basis.
- **Display Management** – manage displays as part of the configuration system. This includes display assignment for download purposes, download status indicators, and version control.
- **Display Distribution** – allows displays to be moved from the configuration system to target nodes.
- **Runtime Structure and Behaviors** – a partial set of the runtime structure is summarized in this document. The requirements stated here are necessary to provide context for the overall configuration and distribution discussions.

1.1 Related documents:

Document:	Controlled copy in Documentation SourceSafe:
Technical Architecture	
Runtime Workspace	

1.2 Objectives

- 1- Fully integrated - Process Displays are fully integrated with the configuration and runtime subsystems.
- 2- Ease of use – the Process Graphics System will be easy to use. Displays will be easy to call up on a wide variety of display devices. No display conversion is required to use displays on different devices. It will be easy to design displays for optimum usability on specific form factors.
- 3- Support best engineering practices. The system will be designed to facilitate maximum re-use of displays and sub-displays through both standard and user-defined

libraries and through display classes, thus minimizing configuration and testing effort.

[Points above are objectives from user's standpoint, below are internal]

- 4- The runtime will be separated from the configuration system. It will be developed independently (different engineering groups separated by 1000's of miles).
- 4-
- 5- Separation of the graphics from the scripting engine. [For example, the graphics and it dynamics are defined in data (xml) that is interpreted by scripts (java)]

1.3 Key Terms

Term	Description
Framework panel	
Runtime workspace	
Workspace application	
Workspace display	
Workspace framework	

2. Requirements

2.1 System Topologies - Single PC, DeltaV Style Systems, and Zones

2.1.1 Single PC

- Requirements from Simulate...

2.1.2 DeltaV Style Systems

- Requirements from single ACN System...

2.1.3 Zones

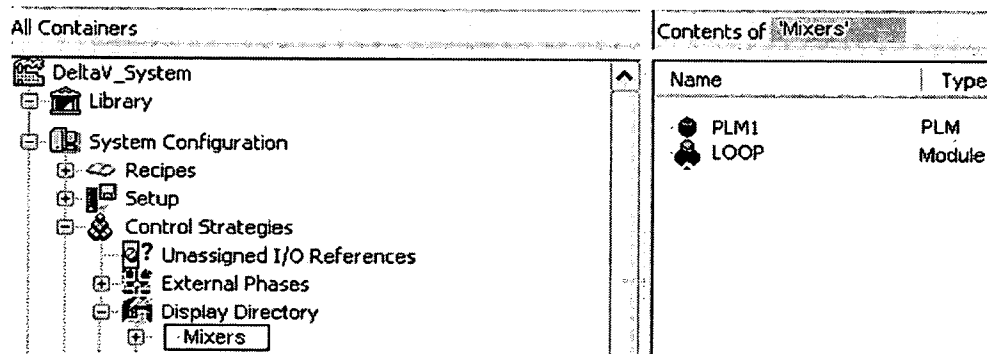
- Requirements from a multiple ACN topology...

2.2 Display Configuration

2.2.1 General

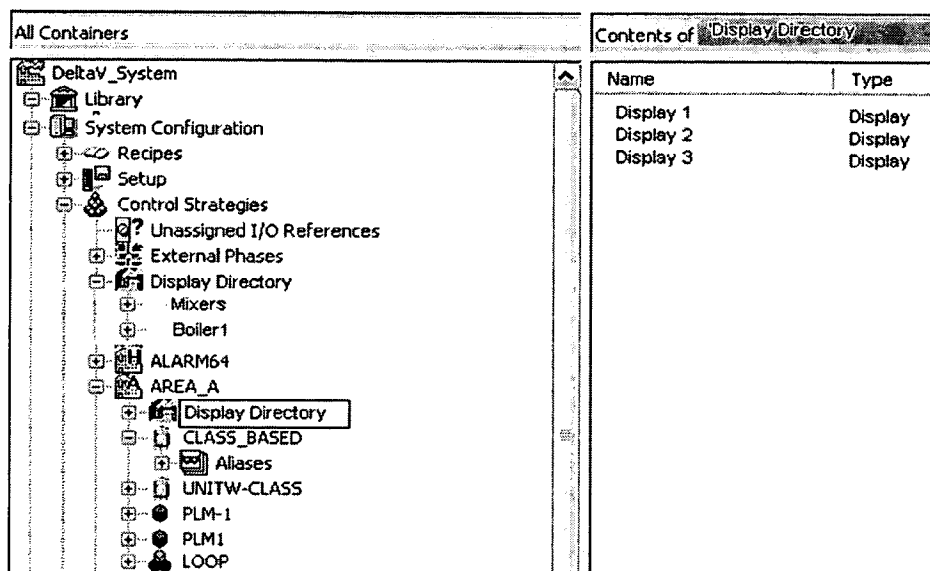
- Support control strategy and hardware renames (e.g. module, function block, parameter).
- Support display reuse for duplicate/similar equipment.
- Provide a mechanism for users to not be forced to send all displays to all workstations.
- Displays are included in the database to allow system integrators to be able to switch databases on the same PC and have the displays switched as well. A total system download should switch out displays in the same way that control strategies are replaced.
- Displays can be assigned to a general display directory. All displays at this level are downloaded to all workstations.

May have display downloads, or are displays may be available through the (possibly redundant) display server. In this case the displays would just be uploaded (published) to the display server. The file locations within the display server would be immaterial since the display server would do its own file management (as occurs in the prototype).



- Displays can also be assigned to areas and follow Area, Unit, Equipment, and Module assignments (covered in more detail below). There are 2 primary reasons why users would move their displays out of the central display directory and into a display directory in the equipment hierarchy:

- Manage which displays are available (because of security) to which workstation.
- Manage displays as part of the item in the equipment hierarchy.
- Users can drag a display to an area, cell, unit or equipment module. Doing so moves the display to the display directory subfolder under that item. Control modules do not have display folders. See below –



- This display is now 'owned' by that item. This means for example, that if the unit is deleted or exported (along with all sub-components), the displays are also deleted or exported.
- Displays can be shared as well as owned (similar to the concept we have of shared components of Module Classes). The same display might be associated with many items of equipment spread across several areas so you would not necessarily want any one item to own it. Display could be owned by an item higher up the hierarchy, or just owned by the library in this case.
- Displays and Display Classes can be associated with Module Classes, with references typically being relative so that they can be automatically resolved at run time or download generation time, requiring no manual input from the user.
- Another major benefit users have with assigning a display to the hierarchy is the 'smart' things that can be done with ownership. Copy of a containing area, cell, unit, EM can perform a copy that allows the references to be automatically changed. For example, UNIT-1 contains 3 modules MOD-1, MOD-2, MOD-3 and DISPLAY1, DISPLAY2. When UNIT-1 is copied, users have the option to provide the new names of the modules and displays, something like:

Current Name	New Name
UNIT-1	UNIT-12
MOD-1	MOD-12
MOD-2	MOD-22
MOD-3	MOD-32
DISPLAY1	DISPLAY12

- In doing this, all references to the 'current name' modules are replaced with the new names. Contained items are shown in the Current Name list. If a module or display had a reference to a module inside another unit, it would not show up here.
- We could also include a mechanism to define 'instance specific' display components that could show up in the equivalent of 'Quick Configure'. For example, perhaps there is a text string that should be changed on each graphic. This is handled by the display parametisation. Any number of top level parameters can be defined for a display. Each parameter can control visual appearance and/or alias substitution within the display.
- When displays assigned to a unit, area, etc., they will be downloaded to workstations that have that area assigned to either the alarms and events subsystem or the operator subsystem. This ensure that if a workstation is expected to get the alarms for an area, it has the displays for that area.
- If a user wants displays to be sent to a workstation that does not have it's area assigned, then that area is added to the workstations operator subsystem.
- A new 'Display Templates' directory is added to the library. Users can place displays in this directory. Copying a display out of the templates directory behaves the same a copying a module. There are display templates which are essentially starting displays, sub-displays which are the equivalent to dynamos today and there are also libraries of display fragments which are just useful reusable bits of displays

2.2.2 Look and Feel

- Uncluttered editing area.
- Frequent operations available without needing to call up menus.
- Multi-level zoom.
- All context operations available from right-click menu.
- WYSIWYG text editing operations.
- Multiple document interface.
- Multiple windows can be open onto the same graphic. Any updates will be reflected in all windows.

2.2.3 Static Elements

- Supports current "best practice" in terms of graphic editing capability.
- Fully support all standard primitive types (line, polyline, rectangle, circle, arc, image, text)
- Support gradient fill for pseudo 3D look to tanks etc.
- Support pipework with 3D look.
- Provide very extensive set of edit functions for manipulating existing items.
- In place editing of grouped objects without requiring ungroup.
- Ability to "drill down" to edit any nested grouping level.
- Ability to add attachment points to items, and then attach pipes or lines to these points. The pipe or line then tracks the item to which it is attached.
- Complex graphic groups can be converted into images which can give better runtime performance.
- User can choose any canvas size for a graphic.

- The user can define named libraries into which they can put arbitrary pieces of displays for later reuse. Each library item is given a name and can be given a description.

2.2.4 Dynamic Elements

- User interaction controls supported, text boxes, sliders, buttons etc.
- Ability to animate a large number of item properties.
- Ability to define user variables as part of a graphic primitive or grouping.
- Variables can be private, for use within the algorithm of an animation.
- Variables can be public and used to parameterize a dynamic algorithm.
- Variables can be "deferred" , i.e. bound to another variable or a process value at an outer scope.
- Item properties which can be animated can be bound to variables or directly to process values.
- Binding to process values can use aliases. i.e. One or more portions of the process value path can be specified as being supplied by the value of a variable (c.f. batch aliasing).
- Variables defined at the "top level" of a display can be used as parameters to the display. When the display is called up the caller can provide values for these parameters.
- Use of variables and aliasing allows a display to be used as a Display Class.
- Animation/scripting capability is provided by function block algorithms which can be attached to any item or group within the display. Function block algorithms read and write user variables.
- Function block diagram editor evaluates the result of the algorithm with every change to the diagram, allowing the configurator to view parameters and values to debug the algorithm without needing to run the display.

2.2.5 Display Storage

- Process displays are stored as part of the overall configuration model. Database support will be added to support
 - References to control items
 - Import/export
 - Versioning
 - Download checking
 - Online upgrades

2.3 Engineering Practices

2.3.1 Display Classes

- Store Display Classes as static/dynamic elements and as a set of Display Parameters. The Display Parameters should be available for bulk edit.
- Process Graphics will invoke the display passing the parameter values defined for the display instance. When invoked the display would be associated with a particular instance of the associated class based item in the equipment hierarchy. Most bindings would be relative to this but there may also be some absolute bindings. A display class simply has a top-level variable which acts as an alias for the module name.

When the display is called up the resolution for this alias is passed as parameter in the display URL.

- Will download the Display and the Display Binding Table to Starburn Process Graphics. May upload to display server.
- Changes to names will automatically be updated in the Display Binding Table and reflected in the download indicators. For bound names, this is similar to bound aliases in module classes.
- Process Graphics will load the Display Class referenced in the Modules and lookup the correct bindings in the Display Parameter List
- Process Graphics will allow for the definition of a Display Class as well as a thumb nail view of the Display Class. This will allow us to create a layered view (e.g. a Unit with Equipment – the equipment is shown as a thumb nail – when selected the equipment view opens up to full view).
- Maintenance displays will be created. These Class-based Displays will be associated with Nodes, Cards, Devices, etc.
- New conditioning monitoring displays will be created. These Class-based Displays will be associated with Condition Monitoring items such as Loop monitoring, Device Alerts, etc.

2.3.2 Display References

- Display item reference holders will be used to link display tags to system related items.
- Any changes to items in the reference holders will cause the binding table(s) download indicators to trip.
- Selecting 'References...' on a display shows any displays referencing the display, and any displays referenced by this display, along with all the data references from the display given the current alias values.
- Selecting 'References...' on a module show any displays referencing the module, in addition to other module references.
-

2.3.3 Bulk Edit

- Bulk Edit of displays...
-

2.4 Display Management

2.4.1 Explorer

- Process Graphics class-based display shortcuts are shown in Explorer in Display Folders (and sub-folders) under System Configuration.
- Process Graphics non-class-based displays are also shown in Explorer in Display Folders (and sub-folders) under System Configuration.
- Process Graphics pictures, either class-based or not, can be assigned to non-class-based modules. Currently Modules contain the following information:
 - a. PRIMARY_CONTROL_DISPLAY
 - b. INSTRUMENT_AREA_DISPLAY
 - c. DETAIL_DISPLAY

These properties will be left as is.

In addition, a list of (Display Type, Display/Display Class) pairs will be added. This way we can generalize the application of e.g. thumbnails, outlines, detail displays, faceplates, maintenance displays, or whatever types make sense. This potentially facilitates the runtime selection of sub-display based on equipment mentioned later.

- The Process Graphics display editor can be launched from Explorer.

2.4.2 VCAT

- A formal interface will be added to handle check in/checkout management tied to Process Graphics. If VCAT is enabled the check-in/check-out versions the displays.
- A DBID will be assigned to Process Graphics displays. This allows users to get VCAT behavior.
- Display Names may be unique across the system. This is because displays are copied into flattened directory structures. This is not necessarily true. Any display hierarchy could be supported if it was felt that it would add value. A flattened structure is certainly the simplest though.

2.4.3 Download indicators

- Manage download indicators on the Operator Subsystem and the Displays Folders. This includes the following: (If we have a display server then most of this goes away)
 - a. The Operator Subsystem on Workstations will be used to provide download indication (i.e. displays need to be downloaded).
 - b. The Display Folder under System Configuration (similar to Recipes) will also be used to provide download indication.
 - c. The download indicator for non-class-based displays is set on a system-wide basis. This means that when a non-class-based display is updated
 - The Operator Subsystem download indicator trips on all workstations.
 - The download indicator on the Display, Display Category, and Displays Folder trips under System Configuration.
 - d. The download indicator for class-based displays is set on a system-wide basis. This means that when a class-based display is updated
 - The Operator Subsystem download indicator trips on all workstations where control modules reference the class-based display.
 - The download indicator on the Display, Display Category, and Display Folder trips under System.
 - e. Update download status will regenerate the download indicator on displays by computing checksums on the displays.

2.5 Display Distribution

2.5.1 Display assignment

- Process Displays follow the Area assignments for Units, Equipment Modules, and Modules on workstations.
- Process Displays are downloaded to the workstations where they are defined. User should be allowed to control exactly which graphics get downloaded to a workstation.
- Process Displays may follow security.

2.5.2 Display downloading

- Context menu of Operator Subsystem includes Download All displays, Download New and Changed Displays.
- Allow users to download all displays from the Displays Folder. This is consistent with a total system download.
- Allow users to download one Display Folder at a time. This is consistent with Recipe Procedure/Unit Procedure/Operation downloads. Note all nodes where the displays are assigned will be downloaded.
- Context menu of a display in the library includes Download and the user can elect to download the display to all workstations to which it applies (which might be all or a subset of the workstations in the system depending on the answer to the earlier question).
- Foreign documents can be associated with both Class-based and non-Class-based displays. This allows the Process Engineers to provide operating instructions associated with Displays that Operators can use to assist in using the item the display is associated with. (Note, if we are going to associate foreign documents with displays, then we should also associate foreign documents with composite templates, module templates/classes so that users can link their custom help information to these objects).

2.6 Runtime Structure and Behaviors

2.6.1 Runtime Structure

- Supports execution in internet explorer.
- Clean separation of display framework from display specifics.
- Graphics defined in SVG file, one per display
- Dynamics defined in JavaScript file, one per display
- Library of support JavaScript is in a single file which can be cached by the client.
- Library of function blocks is in a single file which can be cached by the client.
- Display is invoked through an ASPX page which pulls together the support JavaScript, the function block JavaScript, the display specific SVG file and the display specific JavaScript file.
- The display server and the data server can be separate servers or combined into a single server.

2.6.2 Runtime Behavior

- Graphic can be run by a C# client application or in an internet explorer on any supported platform.
- The internet explorer client windows have no menus or toolbars so that the display application can control navigation and other features of the window.
- C# client uses Web methods to communicate with the data server.
- Internet explorer client uses http post to communicate with the data server.
- Process value bindings can be changed while a display is running, for example by the user changing the value of an alias. This will cause a re-registration with the data server to obtain the new data set.
- One graphic can call another. The calling graphic can dynamically control which display is called up, and the values of any parameters passed to the called display. This enables a Display Class to be invoked on a specific module for example.
- If connection from a graphic to the data server for the display is lost, the connection is automatically re-established when the data server becomes available again.
- The graphic can have independent update periods for running animations and for getting data from the data server.
- Function blocks provide a secure mechanism for defining dynamic behavior which can be readily implemented in both a C# and internet explorer display client.

3. Concept

3.1 Overview

Display management will be very different in Starburn. Unlike DeltaV where Process Displays are edited and managed external to the configuration system, in this system the user always edits the database copy and saves the display directly to the database. In addition, unlike DeltaV where displays are moved around using Yellow-Pages, in displays are downloaded out of the configuration system just as control strategies are. This allows displays to be managed via download indicators and versioned along side their control counterparts.

3.2 Display Configuration

3.2.1 Display Configuration Overview

The graphics system needs to be able to support many constructs including

- save/restore
- downloading
- animation
- real-time data
- parameter substitution (e.g. faceplates and class-based behavior)
- graphic element grouping
- behaviors
- running the display in a Rich or Thin Client
- etc

To illustrate these concepts display 'dynamic' has been created. This display is shown below.

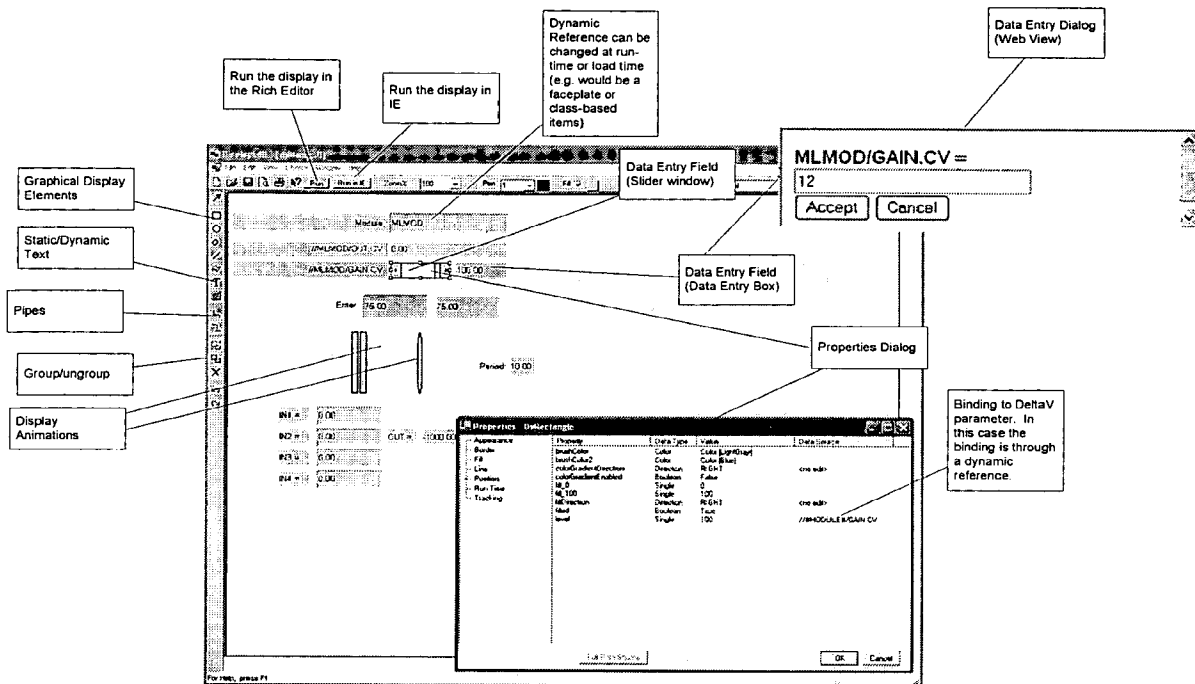


Figure 1. Display 'dynamic'

3.2.2 Display Constructs

3.3 Display Classes

This scenario also inherently covers bulk edit, aliases, and referencing.

3.3.1 Background

This scenario is intended to illustrate the principles of the use of Display Classes. The standard parameter names used are suggestions and subject to review. The plant equipment is shown in the drawing below.

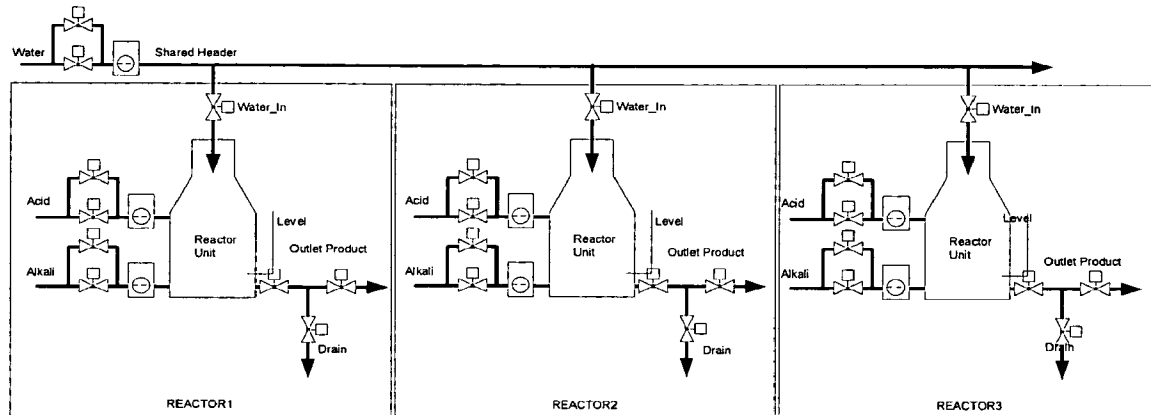


Figure 2. Process Cell - Salts

A plant has a number of reactor units of similar design. In this example there are three units in the process cell named “SALTS”.

Each unit mixes a specified quantity of acid, alkali and water to produce a product. Each reactor has a private feed of acid and alkali but all units share a single water header. Each reactor has an outlet that goes to the product tank or to drain.

In Explorer the configured module classes would appear as:

Library

Advanced Definitions

Control Module Classes

General

+ON_OFF_VALVE
+FLOW_METER
+LEVEL_METER

Equipment Module Classes

General

TOTALIZER	Command-driven
+COARSE_VALVE	ON_OFF_VALVE
+FINE_VALVE	ON_OFF_VALVE
+FLOW_METER	FLOW_METER
+RUN_LOGIC	
+COMMAND_2	Totalize_Accurate
+COMMAND_3	Totalize_Fast
+COMMAND_99	Reset
+STOP_LOGIC etc.	
REACTOR_OUTLET	State-driven
+OUTLET	ON_OFF_VALVE
+DRAIN	ON_OFF_VALVE
+PRODUCT	ON_OFF_VALVE
+RUN_LOGIC	
+STOP_LOGIC etc.	

Unit Classes

General

REACTOR	
DOSE	Phase
MIX	Phase
DRAIN	Phase
FLUSH	Phase
+ACID	TOTALIZER
+ALKALI	TOTALIZER
->WATER	TOTALIZER
+WATER_IN	ON_OFF_VALVE
+LEVEL_METER	LEVEL_METER
+OUTLET	REACTOR_OUTLET

3.3.2 Display Classes

Display Classes are used to make the configuration of displays easier to manage and easier to do. Displays are populated with sub-displays based on a users “Component Level Menu” selection.

3.3.2.1 Process Cell – Display Class “SALTS”

A plant has a number of reactor units of similar design. There are three units named “REACTOR1”, “REACTOR2”, and “REACTOR3” in process cell “SALTS”.

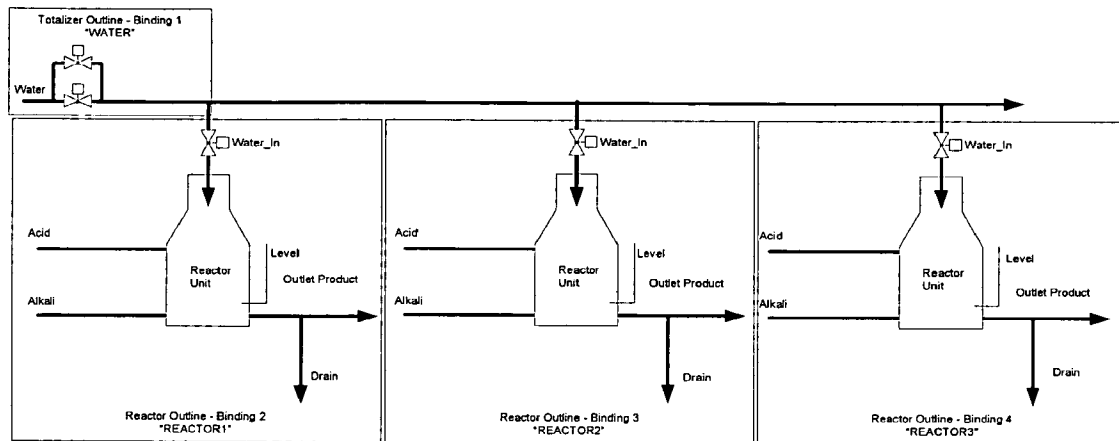


Figure 3. Display Class – “SALTS”

The binding table for the Process Cell Display Class is summarized below:

Binding Table	
1.	TOTALIZER-THUMBNAIL
2.	REACTOR-OUTLINE
3.	REACTOR-OUTLINE
4.	REACTOR-OUTLINE

NOTE : there are actually 3 levels of view of a Totalizer in the examples

- *The one shown above called TOTALIZER-THUMBNAIL (no flow meter).*
- *The one in REACTOR-DETAIL called TOTALIZER-OUTLINE.*
- *The totalizer detail display TOTALIZER-DETAIL.*

3.3.2.2 Unit – Display Class “REACTOR-DETAIL”

The following drawing shows the subdisplays for the Unit Display Class “REACTOR-DETAIL”.

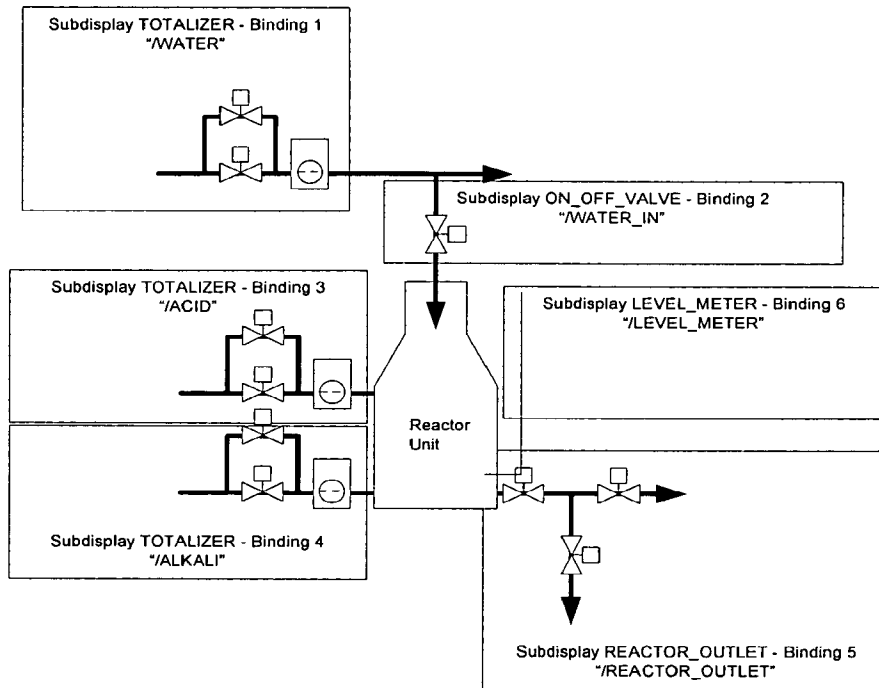


Figure 4. Display Class – “REACTOR-DETAIL”

The binding table is summarized below:

Binding Table	
1.	/WATER
2.	/WATER_IN
3.	/ACID
4.	/ALKALI
5.	/REACTOR_OUTLET
6.	/LEVEL_METER
7.	STATE.CV
8.	/#WATER#/STATE.CV

3.3.2.3 Display Class “TOTALIZER”

The TOTALIZER consists of three control elements, a coarse valve, a fine valve, and a flow meter. These work together to provide a totalizing input capability. From the totalizer display the user can navigate to ON_OFF_VALVE's and FLOW_METER.

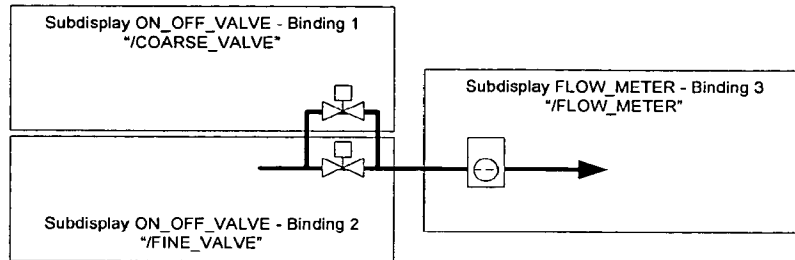


Figure 5. Display Class – “TOTALIZER”

The binding table for the Display Class is summarized below:

Binding Table	
1.	/COARSE_VALVE
2.	/FINE_VALVE
3.	/FLOW_METER

3.3.2.4 Display Class “REACTOR_OUTLET”

The reactor outlet contains three valves which coordinate the transfer of reacted material to drain or to product. From the REACTOR_OUTLET display the user can navigate to ON_OFF_VALVE's.

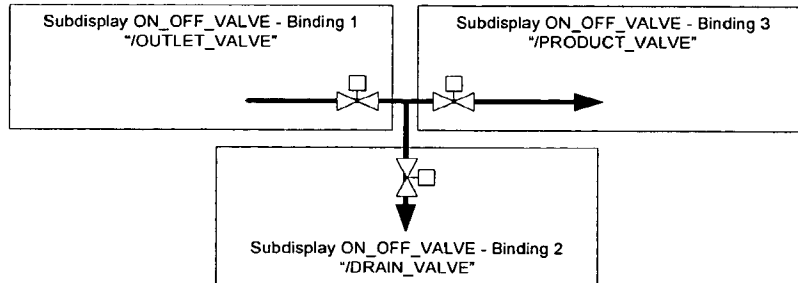


Figure 6. Display Class – “REACTOR_OUTLET”

The binding table for the Display Class is summarized below:

Binding Table	
1.	/OUTLET_VALVE
2.	/DRAIN_VALVE
3.	/PRODUCT_VALVE

3.3.2.5 Display Class “LEVEL_METER”

The LEVEL_METER is a detailed display. There are no further linkages to other displays.

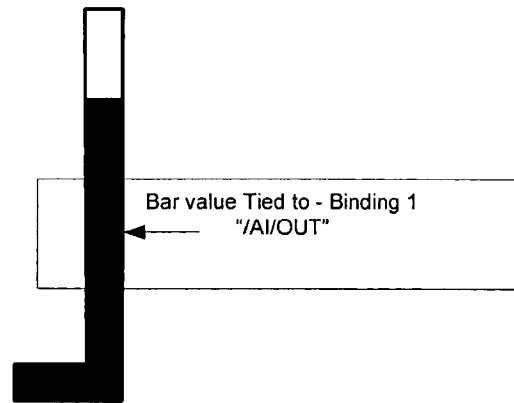


Figure 7. Display Class – “LEVEL_METER”

The binding table for the Display Class is summarized below:

Binding Table	
1.	/AI/OUT
2.	/DRAIN_VALVE
3.	/PRODUCT_VALVE

3.3.2.6 Display Class “ON_OFF_VALVE”

The ON_OFF_VALVE is a detailed display. There are no further linkages to other displays.

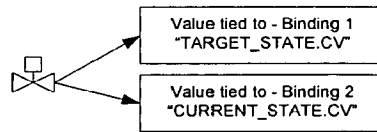


Figure 8. Display Class – “ON_OFF_VALVE”

The binding table for the Display Class is summarized below:

Binding Table	
1.	TARGET_STATE.CV
2.	CURRENT_STATE.CV

3.3.2.7 Display Class “FLOW_METER”

The FLOW_METER is a detailed display. There are no further linkages to other displays.

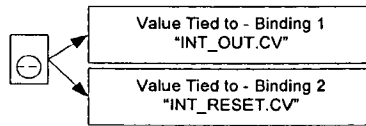


Figure 9. Display Class – “FLOW_METER”

The binding table for the Display Class is summarized below:

Binding Table	
1.	INT_OUT.CV
2.	INT_RESET.CV

3.3.2.8 Summary of Display Class Relationships

The following drawing illustrates the mapping between displays. Some of the displays, such as "TOTALIZER", will have more than one level of detail. In the case of TOTALIZER we have "TOTALIZER-THUMBNAIL", "TOTALIZER-OUTLINE", and "TOTALIZER-DETAIL".

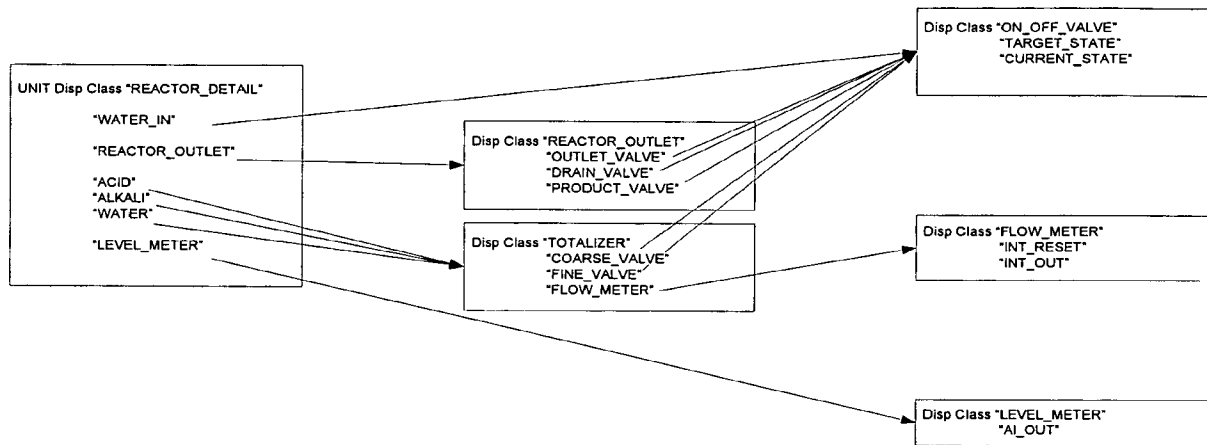


Figure 10. Display Class Relationships

3.3.3 Display Instances

3.3.3.1 Download Bindings

The following summarizes what the user would see in the Library hierarchy.

NOTE : We would not see instance information here. The display class would have short-cuts to sub-display classes used. This is information that we would extract from the display and store as references so we know the dependencies. A top level library category for display classes will be added along with sub-categories for organizing. Short-cuts have been shown with ->.

```
Library
  Display Classes
    Unit Displays
      REACTOR_DETAIL
        ->TOTALIZER_OUTLINE
        ->REACTOR_OUTLET_OUTLINE
        ->ON_OFF_VALVE_OUTLINE
        ->LEVEL_METER_OUTLINE
      :
    Equipment Module Displays
      TOTALIZER
        ->ON_OFF_VALVE_OUTLINE
        ->FLOW_METER_OUTLINE
      :
      REACTOR_OUTLET
        ->ON_OFF_VALVE_OUTLINE
    Control Module Displays
      FLOW_METER_DETAIL
      FLOW_METER_OUTLINE
      LEVEL_METER_DETAIL
      LEVEL_METER_OUTLINE
      ON_OFF_VALVE_DETAIL
      ON_OFF_VALVE_OUTLINE
```

3.3.3.2 Instance Tree

The following summarizes what user would see in the Control Strategies hierarchy.

```
Process Cell "SALTS"
  CBCM "WATER"
    CBCM "COARSE_VALVE_1"
    CBCM "FINE_VALVE_1"
    CBCM "FLOW_METER_1"
  CB UNIT "REACTOR1"
    Shortcut to "WATER"
    CBEM "ACID_1"
      CBCM "COARSE_VALVE_1_1"
      CBCM "FINE_VALVE_1_1"
      CBCM "FLOW_METER_1_1"
    CBEM "ALKALI_1"
      CBCM "COARSE_VALVE_1_2"
      CBCM "FINE_VALVE_1_2"
      CBCM "FLOW_METER_1_2"
    CBEM "REACTOR_OUTLET_1"
      CBCM "OUTLET_VALVE_1"
      CBCM "DRAIN_VALVE_1"
      CBCM "PRODUCT_METER_1"
    CBCM "WATER_IN_1"
    CBCM "LEVEL_METER_1"
  CB UNIT "REACTOR1"
  :
  CB UNIT "REACTOR1"
  :
```

3.3.4 Display Class Downloads

The display system will hold the following:

- A list of items in equipment hierarchy, each having a list of (Display Type and Display/Display Class) pairs.
- A display having
 - A binding list which is simply a list of paths, which may be absolute or relative to the current equipment item.
 - A display in which
 - Sub-display elements are either statically bound to Displays/Display Classes or are bound to a Display Type which will be used to determine the actual display from the bound equipment item.
 - All dynamic elements, including sub-displays, are cross referenced to an entry in the binding table.

3.4 Display Management and Distribution

3.4.1 Displays Folder and Sub-folders

A Displays Folders will be added to the Explorer under System Configuration. The Displays Folder will support Display Categories. Users can arrange their Class-based as well as their non-Class-based displays in the displays folders.

3.4.2 Operator Subsystem

The Operator Subsystem under workstations will be used to indicate whether or not displays require downloading. When set it indicates that displays need to be downloaded.

3.4.3 Download

There are two locations to download displays from. The most general is the Operator Subsystem on Workstations. When the user downloads from here all they know is that all displays, including display classes, will be downloaded. The second location to download displays from is the Displays Folder under System configuration.

When Users download displays from the 'Displays Folder' all workstations in the system are downloaded. The User may also select the Displays Subfolder. When they download from here all of the workstations where displays hold associations will be downloaded. The user may also drop down to specific displays and download them one at a time. When they do this all workstations where the display holds an association will be downloaded.

3.5 Interface with the Runtime

3.5.1 Display Downloads

Supporting both Rich (Win Forms) and Thin Clients (IE, Handhelds, Smart Phones) requires the cooperation of several techniques. Rich Clients can directly load the native SVG format. Thin Clients make use of a combination of SGV and Java scripting. To illustrate these ideas Process Display 'DYNAMIC' is used below. Associated with Process Display 'DYNAMIC' are the following files:

dynamic.SVG/XMAL	The Rich Client files will be in XML format.
OpenDisplay.aspx	Initial HTML file – used to select display to be displayed.
OpenDisplay.js	Works with OpenDisplay.aspx to open initial display.
FunctionBlocks.js	Provides display element dynamic behavior.
dynamic.dvg.SVG/XMAL	SVG/XMAL format of dynamic.SVG/XMAL file.
dynamic.dvg.js	Provides dynamic [i.e. scripting] behavior for display 'dynamic.SVG/XMAL'

The relationship of the files is illustrated and discussed below.

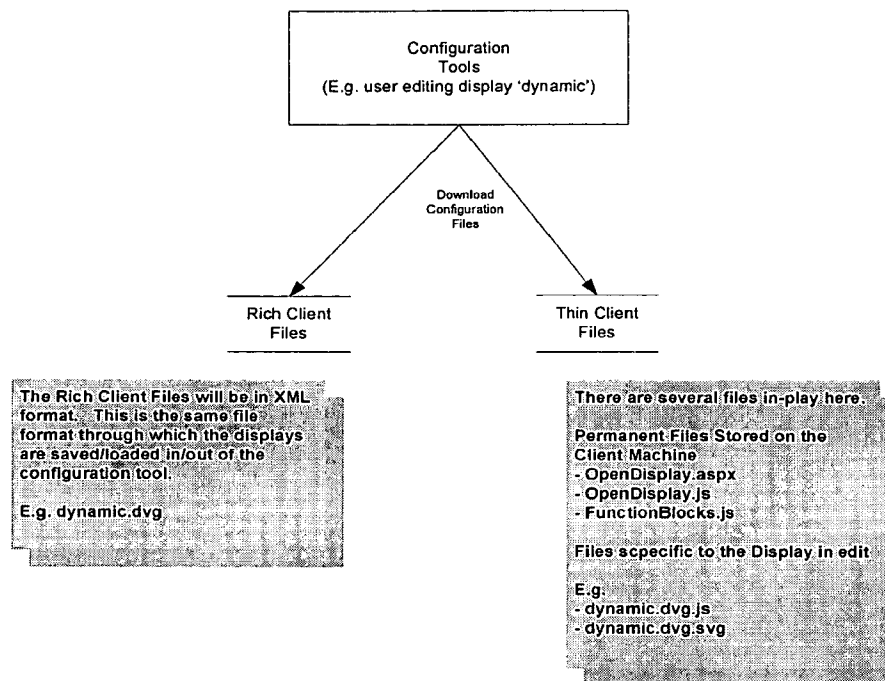


Figure 11. Display Storage

3.5.1.1 Rich Client

3.5.1.1.1 dynamic.dvg

The file 'dynamic.dvg' contains the complete data storage for the display. Both the editor and the rich client for runtime access this file.

3.5.1.2 Thin Client

3.5.1.2.1 OpenDisplay.aspx & OpenDisplay.js

The file 'OpenDisplay.aspx' is used to open the thin client display. The file provides several functions:

- opens the display requested by the user
- using 'OpenDisplay.js' identifies all of the dynamic fields and registers for updates
- handles dynamic updates
- sets up the default data entry dialog
- handles user entered data

This file 'OpenDisplay.aspx' is listed below:

```

<%@ Page language="c#" Codebehind="OpenDisplay.aspx.cs" AutoEventWireup="false" Inherits="WebUI.OpenDisplay" %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN" >
<HTML>
  <HEAD>
    <TITLE></TITLE>
  
```

```

<META NAME="GENERATOR" Content="Microsoft Visual Studio 7.0">
<script language="javascript" src="<% jsName(); %>"></script>
<script language="javascript" src="OpenDisplay.js"></script>
<script language="javascript" src="FunctionBlocks.js"></script>
</HEAD>
<BODY onload="initPage();" onunload="deregisterForPageUpdates()" style="MARGIN: 0px">
  <EMBED id="id_Graphic" src="<% displayName(); %>" type="image/SVG/XMAL+xml" height="800"
width="600" pluginspage="http://www.adobe.com/SVG/XMAL/viewer/install/"></EMBED>
  <p id="id_resyncCount"></p>
  <p id="id_nClient"></p>
  <form>
    <p><INPUT id="reset" type="button" onClick="remoteAction('Reset');" value="Reset Server"
name="reset"></p>
  </form>
  <div id="id_enterValueDiv" style="display: none">
    <iframe id="id_enterValueFrame" src="EnterValue.htm" style="position: absolute; top: 100; left: 100;
height: 90; width: 350; background-color: #999999 ; border: 1px solid black" border="0"></iframe>
  </div>
</BODY>
</HTML>

```

The file 'OpenDisplay.js' contains many pre-built functions. These functions scan the 'SVG/XMAL' file identifying parameters that need to register for dynamic updates, handle dynamic updates, handle user entered values, etc. A small part of this file is listed below.

```

//-----
// Constructor for object holding dynamic element information
//-----
function DynamicElement( node , script , nodeAttribute , valueConditioning , tag )
{
  this.node = node ;
  this.nodeAttribute = nodeAttribute ;
  this.valueConditioning = valueConditioning ;
  this.tag = tag ;
  this.script = script ;
}

//-----
// Function to capture the SVG/XMALDocument to that we can manipulate it from
// our script
//-----
function loadSVG/XMAL( evt )
{
  var SVG/XMALDoc = evt.getTarget().getOwnerDocument() ;
  var SVG/XMAL = SVG/XMALDoc.getElementById( "id_SVG/XMAL" );

  var embed = document.getElementById("id_Graphic");
  embed.setAttribute( "width" , SVG/XMAL.getAttribute("width") );
  embed.setAttribute( "height" , SVG/XMAL.getAttribute("height") );

  SVG/XMALDocument=SVG/XMALDoc ;
}

//-----
// Gets the list of tags which are referenced in the SVG/XMAL document
// Keeps a list of dynamic element objects which should be updated
//-----
function getListOfTagRequests()
{
  tagIndices = new Array() ;
  :
}

//-----
// Prompts the user to enter a new value for a tag
//-----

function enterValue( tag , x , y )
{
  // Move to just below the entry field
  id_enterValueFrame.window.frameElement.style.left = x ;
  id_enterValueFrame.window.frameElement.style.top = y+30 ;
  :
}

```

```
}
```

3.5.1.2.2 FunctionBlocks.js

The file 'FunctionBlocks.js' is used to support the addition of scripting behavior. Scripting behavior allows the user to perform additional data manipulations such as scaling a parameter, manipulating parameters, performing logic and taking action based on the logic, etc². A small part of the file is listed below:

```
function ADD4( index , IN1 , IN2 , IN3 , IN4 )
{
    this["ADD4_OUT_"+index] = -(IN1 - IN2 - IN3 - IN4) ;
}
function DIV( index , IN , DVR )
{
    this["DIV_OUT_"+index] = IN/DVR ;
}
function SEL ( index , IN , VT , VF )
{
    if( IN != 0 )
    {
        this["SEL_OUT_"+index] = VT ;
    }
    else
    {
        this["SEL_OUT_"+index] = VF ;
    }
}
function RD ( index , tag )
{
    this["RD__"+index] = valueOf( tag );
}

function WD ( index , tag , value )
{
    updateValue( tag , value );
}
```

3.5.1.2.3 dynamic.dvg.SVG/XMAL

The file 'dynamic.dvg.SVG/XMAL' is the SVG/XMAL version of the file 'dynamic.dvg'. Dynamic data behaviors have been factored out into java scripts.

```
<?xml version="1.0" encoding="utf-8" standalone="yes"?>
<SVG/XMAL xmlns:dv="abcdefg" id="id_SVG/XMAL" width="1024" height="768" onload="loadSVG/XMAL(evt);">
  <g dv:type="canvas" dv:update-period="100">
    <rect width="1024" height="768" fill="#ebebeb" />
    <g dv:type="none">
      <rect x="2" y="2" width="1020" height="764" fill="none" stroke="#000000" stroke-width="1" />
    </g>
  </g>
  <g>
    <rect x="250" y="35" width="175" height="30" fill="#d3d3d3" onclick="enterValue('__MODULE_1',250,35);"
    onmouseover="eventMouseOver(evt)" onmouseout="eventMouseOut(evt)" />
    <text x="250" y="54" fill="#000000" font-family="Arial" font-size="10pt" text-anchor="start"
    onclick="enterValue('__MODULE_1',250,35);" onmouseover="eventMouseOver(evt)"
    onmouseout="eventMouseOut(evt)">MLMOD<dv:dynamics id="id_DYNAMIC0" tag="__MODULE_1" attribute="text" /></text>
    <rect x="10" y="35" width="235" height="30" fill="#d3d3d3" />
    <text x="245" y="54" fill="#000000" font-family="Arial" font-size="10pt" text-anchor="end">Module:</text>
    <rect x="10" y="80" width="235" height="25" fill="#d3d3d3" />
    <text x="245" y="96" fill="#000000" font-family="Arial" font-size="10pt" text-anchor="end">//MLMOD/OUT.CV<dv:dynamics
    id="id_DYNAMIC1" tag=""+__MODULE_1+"/OUT.CV" attribute="text" valueConditioning="val="+__MODULE_1+"/OUT.CV"
    /></text>
```

² These behaviors were supported in iFIX using VBScripting and in FIX using custom FIX logic. Rich clients can directly execute the behaviors. Thin Clients get exactly the same behavior support using Java scripting.

```

<rect x="250" y="80" width="175" height="25" fill="#d3d3d3" />
<text x="250" y="96" fill="#000000" font-family="Arial" font-size="10pt" text-anchor="start">0.00<dv:dynamics id="id_DYNAMIC2"
tag="" + __MODULE_1 + "/OUT.CV" attribute="text" valueConditioning="val=fmt(val,2)" /></text>
<rect x="250" y="115" width="90" height="25" fill="#d3d3d3" stroke-width="0">
  <dv:dynamics id="id_DYNAMIC3" tag="" + __MODULE_1 + "/GAIN.CV" attribute="width" valueConditioning="val=(90*val/100.0)"
/>
:
  <g>
    <dv:dynamics id="id_DYNAMIC7" script="dynamic_7(SVG/XMALContext);" />
    <rect x="140" y="355" width="95" height="25" fill="#d3d3d3" onclick="enterValue('_IN1_4',140,355);"
onmouseover="eventMouseOver(evt)" onmouseout="eventMouseOut(evt)" />
  :

```

3.5.1.2.4 dynamic.dvg.js

The file 'dynamic.dvg.js'

```

function dynamic_7(SVG/XMALContext)
{
  RD(0,'_IN1_4');
  RD(1,'_IN2_6');
  RD(2,'_IN3_5');
  RD(3,'_IN4_3');
  ADD4(4,RD__0,RD__1,RD__2,RD__3);
  WD(5,'_OUT_8',ADD4_OUT_4);
}

```

Display Logic and Language

1.	STARBURN-DISPLAY LOGIC AND LANGUAGE CONCEPT.....	466460
1.1	OBJECTIVES	466460
1.2	KEY TERMS.....	468462
1.3	TYPES OF DISPLAY LOGIC.....	468462
1.3.1	<i>Framework Logic</i>	468462
1.3.2	<i>Custom Logic</i>	469463
1.4	LOGIC AND LANGUAGE CONCEPTS & USER NEEDS	469463
1.4.1	<i>Graphics and Logic Separation</i>	469463
1.4.1.1	Concept	469463
1.4.1.2	User Needs.....	470464
1.4.2	<i>Logic Execution Environments</i>	471464
1.4.2.1	Concept	471464
1.4.2.2	User Needs	471464
1.4.3	<i>Multiple Display Device Support</i>	471465
1.4.3.1	Concept	471465
1.4.3.2	User Needs	472465
1.4.4	<i>Dynamic Behaviors, Interactions, & Animations</i>	473466
1.4.4.1	Concept	473466
1.4.4.2	User Needs.....	473466
1.4.5	<i>Behavior Classes or Scriptlets</i>	474467
1.4.5.1	Concept	474467
1.4.5.2	User Needs.....	475467
1.4.6	<i>Display Logic Configuration</i>	475468
1.4.6.1	Smart Dialog and Property-Based Configuration.....	475468
1.4.6.1.1	Concept	475468
1.4.6.1.2	User Needs.....	476468
1.4.6.2	Function Block-Based Logic Configuration.....	477469
1.4.6.2.1	Concept	477469
1.4.6.2.2	User Needs.....	477469
1.4.6.3	Script-Based Logic Configuration	477470
1.4.6.3.1	Concept	477470
1.4.6.3.2	User Needs	477470
1.4.7	<i>Inter and Intra Display Communications</i>	478470
1.4.7.1	Display Variables and Contexts.....	478470
1.4.7.1.1	Concept	478470
1.4.7.1.2	User Needs.....	458471
1.4.7.2	Inter-Display Component Communications	458471
1.4.7.2.1	Concept	458471
1.4.7.2.2	User Needs.....	459472
1.4.8	<i>Display and Component Persistence</i>	460472
1.4.8.1	Session Persistence – Component Scratchpad.....	460472
1.4.8.1.1	Concept	460472
1.4.8.1.2	User Needs.....	461473
1.4.8.2	Long-Term Persistence	461473
1.4.8.2.1	Concept	461473
1.4.8.2.2	User Needs.....	462473
1.4.9	<i>Debugging and Simulation</i>	462474
1.4.9.1	Concept	462474
1.4.9.2	User Needs.....	463474

<i>1.4.10 Migration and Versioning.....</i>	<i>463475</i>
1.4.10.1 Concept.....	463475
1.1.1.1 User Needs.....	463475
1.4.10.2.....	463475
<i>1.4.11 Version Control & Proprietary Information Protection.....</i>	<i>464475</i>
1.4.11.1 Concept.....	464475
1.4.11.2 User Needs.....	464475

<u>1.</u>	<u>Starburn Display Logic and Language Concept</u>	<u>3</u>
<u>1.1</u>	<u>Objectives</u>	<u>3</u>
1.2	Logic and Language Concepts & User Needs	4
1.2.1	Graphics and Logic Separation	4
1.2.1.1	Concept	4
1.2.1.2	User Needs	5
1.2.2	Logic Execution Environments	5
1.2.2.1	Concept	5
1.2.2.2	User Needs	6
1.2.3	Multiple Display Device Support	6
1.2.3.1	Concept	6
1.2.3.2	User Needs	6
1.2.4	Dynamic Behaviors, Interactions, & Animations	7
1.2.4.1	Concept	7
1.2.4.2	User Needs	7
1.2.5	Behavior Classes - Scriptlets	8
1.2.5.1	Concept	8
1.2.5.2	User Needs	9
1.2.6	Display Logic Configuration	9
<u>1.2.6.1</u>	<u>Smart Dialog and Property-Based Configuration</u>	<u>9</u>
<u>1.2.6.1.1</u>	<u>Concept</u>	<u>9</u>
<u>1.2.6.1.2</u>	<u>User Needs</u>	<u>10</u>

<u>1.2.6.2</u>	<u>Function Block-Based Logic Configuration</u>	<u>11</u>
<u>1.2.6.2.1</u>	<u>Concept</u>	<u>11</u>
<u>1.2.6.2.2</u>	<u>User Needs</u>	<u>11</u>
<u>1.2.6.3</u>	<u>Script-Based Logic Configuration</u>	<u>11</u>
<u>1.2.6.3.1</u>	<u>Concept</u>	<u>11</u>
<u>1.2.6.3.2</u>	<u>User Needs</u>	<u>11</u>
<u>1.2.7</u>	<u>Inter and Intra Display Communications</u>	<u>12</u>
<u>1.2.7.1</u>	<u>Display Variables and Contexts</u>	<u>12</u>
<u>1.2.7.1.1</u>	<u>Concept</u>	<u>12</u>
<u>1.2.7.1.2</u>	<u>User Needs</u>	<u>12</u>
<u>1.2.7.2</u>	<u>Inter-Display Component Communications</u>	<u>12</u>
<u>1.2.7.2.1</u>	<u>Concept</u>	<u>12</u>
<u>1.2.7.2.2</u>	<u>User Needs</u>	<u>13</u>
<u>1.2.8</u>	<u>Display and Component Persistence</u>	<u>14</u>
<u>1.2.8.1</u>	<u>Session Persistence – Component Scratchpad</u>	<u>14</u>
<u>1.2.8.1.1</u>	<u>Concept</u>	<u>14</u>
<u>1.2.8.1.2</u>	<u>User Needs</u>	<u>14</u>
<u>1.2.8.2</u>	<u>Long-Term Persistence</u>	<u>15</u>
<u>1.2.8.2.1</u>	<u>Concept</u>	<u>15</u>
<u>1.2.8.2.2</u>	<u>User Needs</u>	<u>15</u>
<u>1.2.9</u>	<u>Debugging and Simulation</u>	<u>15</u>
<u>1.2.9.1</u>	<u>Concept</u>	<u>15</u>

_1.2	Dynamic Behaviors	3
_1.2.1	User Needs	3
_1.3	Display Logic Configuration	4
_1.3.1	User Needs: Dialog-Based Configuration	4
_1.3.2	User Needs: Script-Based Configuration	5
_1.4	Inter and Intra Display Component Communication	5
_1.4.1	User Needs	6
_1.5	Component ScratchPad and Mailboxes	6
_1.5.1	Concept: Component Scratchpad	6
_1.5.2	User Needs: Component Scratchpad	6
_1.5.3	Concept: Component Mailboxes	6
_1.5.4	User Needs: Component Mailboxes	7
_1.6	Component Persistence	7
_1.7	Behavior Classes and Groups	7
_1.8	Scripting Execution Environment	7
_1.9	Debugging and Simulation	7
_1.10	Display Form Factors and Logic Implications	7
_1.11	Migration and Versioning	7
=		
=		
=		
==		



==

==

==

==

==

==

==

==

==

==

==

==

==

==

==

==

==

==

==

==

==

==

=

=

=

=

=

1. ~~Starburn~~ DDisplay Logic and Language Concept

This document describes the concept for display logic and language support as it pertains to the runtime operator's workspace within the ~~Starburn~~ development program. Its goal is to identify user needs in this area, and to define concepts for a collection of system features that would meet those needs. Toward that end, this concept document defines the primary methods and means by which operational displays and the objects contained within them can be manipulated and linked together in order support dynamic, animated behaviors.

1.1 Objectives

The primary objective for the ~~Starburn~~ display logic and language feature set is to provide end users the capability to configure and engineer operator displays and display components which dynamically interact with the underlying control system (including runtime data, historical information, field devices, etc...). In the majority of cases, this configuration effort should require little to no programming effort, but the environment may maintain the ability to customize pre-defined functionality through scripting to meet advanced application requirements. ~~Some~~ The primary of the major goals for the logic and language sub-system and supporting workspace functions are:

1. Provide tools to allow users to easily engineer and configure dynamic displays that interact with the underlying process control system, plant personnel, and other display components. These tools should provide access to all component properties without requiring the use of programming or scripting (unless desired).
2. Provide a logic configuration subsystem that clearly separates the graphical display elements from any underlying animation logic that controls the dynamic behaviors.
3. Clearly separate display logic and dynamic behaviors from the underlying runtime components and services. This is accomplished by not directly accessing any runtime libraries or APIs from the display logic, but rather utilizing intermediate services to collect, write, and react to the underlying process data.
4. Leverage language and logic technologies that work equally well across a multitude of display devices without the need for any modification or loss of functionality (single or multi-monitor displays, PDAs, Cellular Phones, etc...).

5. Leverage drag-and-drop technologies as well as wizards and other graphical-based tools to automatically generate the code behind the dynamic behavior.
6. Provide pre-defined/built-in *“behavior classes”* and *“function blocks”* for the most commonly utilized dynamic behaviors (e.g. color or visibility changes based on field device state).
7. Leverage Starburn’s smart objects and links to forecast or predict the most likely wanted behaviors and provide wizards to minimize configuration effort (e.g. pump and valve components automatically show consistent dynamic behavior if connected by a link).
8. Allow users to name or “alias” components (both class and instance) within or across displays so that both intra and inter-display component communication can take place. This includes allowing both local and global scope attributes or parameters to be defined and exposed on displays or display elements, which can then be used by the display logic to store temporary calculated values or pass information between display elements.
9. Provide tools to view and or edit the script behind the dynamic behaviors if required.
10. Allow for off-line simulation and debugging, including step-by-step logic execution and tracing of the underlying script.

1.2 Key Terms

Term	Description
<u>Display Logic</u>	Programmable script or xml associated with a <u>Starburn</u> graphic intended to provide displays with dynamic behaviors and interactions .
<u>Framework Logic</u>	Display logic , which is not customizable by end-users, and is largely shared across displays so that they function in a consistent and expected manner.
<u>Custom Logic</u>	Display logic , which is intended for customization by end-users, to provide each display with unique characteristics and behaviors at runtime.
<u>Dynamic Behaviors</u>	Properties and methods associated with graphical display elements that provide for the elements to take runtime interactions and animations based on user and process events.
<u>Process Dynamic</u>	A field input or output that can be assigned to a display element such that the underlying display runtime will automatically establish subscription-based updates on behalf of the display.
<u>Runtime workspace</u>	The interactive DeltaV application(s) that provide the <u>Starburn</u> operator interface environment.
<u>Workspace framework</u>	The configurable layout of the runtime workspace ; how it will appear on display hardware.

¹ If situations arise where customization is required in this area, simple hooks or extensions can be added to the custom logic, so that again, the framework logic is tightly controlled and protected.

1.3 Types of Display Logic

There are two basic and distinct types of logic that will ultimately make up a Starburn graphic's display logic. The first is **framework logic**. The second is **custom logic**.

1.3.1 Framework Logic

Framework logic describes all of the logic and script associated with a display that is core to the display's ability to function correctly within the overall Starburn-system. This type of logic will likely be boilerplate in nature, will in some cases be shared across multiple displays, and will be hidden and unchangeable from the user's perspective¹. In this sense, the framework logic can be thought of more as a shared set of "library" routines that provide the basic functionality for all displays to interact with the Starburn-workspace, underlying process control system, and one another.

An example of framework logic might be a well-known, global-scope, JavaScript function—**OpenDisplay**—that can be called by any display component to request another display to be opened. Another example of this type of logic would be the framework routine responsible for determining what specific control system values are required by a given display at runtime in order for the display to function as expected. This routine, or one similar to it, might also be responsible for establishing the underlying connections and subscriptions for the display so that real-time updates occur seamlessly. In either case, no customer or end-

¹ If situations arise where customization is required in this area, simple hooks or extensions can be added to the custom logic, so that again, the framework logic is tightly controlled and protected.

user editing or modifications would be allowed to this type of logic. Instead, the overall logic and language subsystem would be responsible for generating this type of code whenever needed.

1.3.2 Custom Logic

The second type of logic, which is the main focus of this document, is custom logic. This type of display code is specifically targeted for direct end-user manipulation (either through configuration or direct editing), such that very specific and custom-interactions can be defined for a given display. This type of logic would tend to make a display unique from others and would likely be used to drive the overall dynamic behavior of the display at runtime. In other words, custom logic would work in conjunction with framework logic by way of receiving (or sending) information from the framework logic, but then acting on that information in unique, non-shared way.

Custom logic will likely be saved and edited in a one-to-one relationship with a display and will not be shared between displays¹.

1.4 Logic and Language Concepts & User Needs

1.4.1 Graphics and Logic Separation

1.4.1.1 Concept

One of the more important concepts that may be maintained throughout the entire Starburn development is the notion of separating the process control and process graphics. Part of meeting this requirement is to restrict the underlying display logic from making direct use or calls to the Starburn-provided core-runtime services. Instead, the display logic will rely on intermediate runtime-to-display conversion services and utilities to enable graphical depiction and control of the plant processes. This concept is discussed thoroughly in the- Starburn Process Graphics Edit, Save, Download, Run General Scenarios document.

Another requirement of this important separation concept that directly affects the language and logic subsystem is breaking the display's graphical elements, a.k.a-i.e. User Interface (UI) components and any accompanying programmatic logic, into separate physical storage mediums². Separating the display components and associated logic allows for easier distribution, versioning control, and overall management of the distinct parts that make up a Starburn-graphic (of course in reality, the files work in tandem and while they may load and work in a limited fashion without the accompanying sister files, very little of the overall expected behavior will function correctly). Having code and UI components kept separately also promotes good programmatic constructs like abstraction, encapsulation, and code re-use, where both SVG-based and JavaScript-based "components" can be combined to build yet more re-usable modules.

¹ With the special exception of display classes, where the majority of the custom logic would be "shared" across instances, and only a small piece of logic (e.g. alias resolution script) would be truly unique.

² Displays will likely be more of a "logical" concept vs. a physical file, where the "display" is made up of potentially several files, some which include graphical elements (SVG, XHTML, images, etc...) and some including logic (JavaScript).

1.4.1.2 User Needs

- a.) The concept of logic and graphic elements may be transparent to the user at both configuration and run-time. Regardless of how the underlying display components are organized, from the user's perspective the graphic should feel and maintain the appearance of being a single entity¹.
- b.) User's will want the ability to quickly test and debug the display logic and its interaction with the graphical components.
 - should not require full-saves and "publishing" of the graphic and logic for test
 - relatively "fast" switch between design and test mode
 - may present user with accurate prediction of runtime behavior

~~Dynamic Behaviors~~

~~Dynamic behaviors are the basic properties exposed and actions taken by the various components within a display graphic. These behaviors provide the mechanism(s) by which displays take on the dynamic look and feel based on how these behaviors have been configured and driven by the underlying display logic.~~

~~User Needs~~

~~Users will typically expect the behaviors of individual components to follow similar configuration requirements as do other components throughout the Starburn system (e.g. SFCs, Function Blocks, Equipment Modules, etc.). This means that, in general, configuration of the behaviors should be capable of being accomplished via dialog-based properties and browse-for-path-type operations.~~

~~There are likely to be a substantial amount of commonly utilized behaviors and actions, which should be pre-defined based on "behavior classes". An example would be grouping several components together, all of which change color or visibility based on the same field device trigger.~~

~~Users will also typically want to have "buddy" components coordinate their behaviors based on common states or triggers, and Starburn's smart objects and links should provide avenues in which to aid this configuration greatly.~~

~~The following is a non-exhaustive list of some the more commonly used dynamic behaviors:~~

- ~~Visibility (hide/show, flashing)~~
- ~~Movement/Animation~~
- ~~Enabling/Disabling~~
- ~~Color~~
- ~~Resizing~~
- ~~Text or Caption Changes~~

¹ Albeit with possibly multiple views, e.g. "WYSIWYG", "logic configuration", and "code" views.

State Changes (depressed buttons, switches on/off)
Picture/Bitmap Changes
...

1.4.2 Logic Execution Environments

1.4.2.1 Concept

Another important concept associated with the Starburn graphical language and logic subsystem is that it be separated from any particular display device or form factor. In theory this should mean any display configured for one type of device, say a single monitor, should at a minimum be “loadable” onto another type of device, say a PDA, without any re-work. In reality, due to screen real estate and scaling issues some graphics will not be practical for use on certain devices without first being designed for dual purposes. The key point, however, is that no display logic component can require a particular type of display device in order to present the desired behavior for which it was intended. In general, the technologies targeted for the Starburn-graphics subsystem, xml, svg, ~~xhtml~~, javascript, etc..., will go a long way intoward meeting this requirement without a tremendous amount of engineering effort¹.

With this concept in mind, the actual display execution environment will most likely be 3rd party SVG viewers, JavaScript interpreters, and HTML browsers (e.g. Internet Explorer, Adobe SVG Viewer, etc...). Whether or not the Starburn Runtime Operator Workspace leverages such a 3rd party plug-ins or internally executes the display content in some other fashion optimized for single and multiple monitor based systems is immaterial—the important concept is that both “rich” and “thin” clients will process the same display files and utilize an intermediary display server for both distribution and runtime interfacing between display and control system dynamicsvariables.

1.4.2.2 User Needs

- a.) Users should not require any knowledge of the underlying execution environment when configuring displays and its associated logic.
- b.) Users will need some way to “test” out any display logic being developed in a variety of execution environments. This means it will be necessary to emulate as many of the targeted platforms as possible on the development machine, without forcing the user actually have all possible devices present during configuration.

1.4.3 Multiple Display Device Support

1.4.3.1 Concept

Starburn’s overall architecture is targeted for many types of displays devices and even multiple control system I/O (at least DeltaV and Ovation, but possibly more). These requirements dictate using two driving principles in all graphic logic: first, separation of display from runtime services, and second, utilizing technologies that have no affinity toward any given type of display device (see Sections 1.24.1 and 1.24.2). Maybe moreJust as important than simplyas using ubiquitous technologies, the display logic may also be aware and take advantage of the overall **runtime workspace** framework (parent panel and

¹ Its also conceivable that certain Microsoft technologies, e.g. .Net Remoting, may also be available under this constraint as long as all target platforms will function as expected with any such tools.

sizes, anchor points, automatic scaling, aspect ratios, etc...) and automatically decide on the “best fit” for a particular display device. This will mean ~~and~~ re-orienting and re-scaling its component to meet the device’s capabilities at runtime.

In order to accomplish this, each display will likely require a “best fit” set of instructions (e.g. possibly held in an accompanying xml-stylesheet file), which is generated by the configuration system based on the overall ~~runtime operation’s workspace framework~~ and layout for a given devices(s). This file should aid the display’s embedded components and associated logic to determine information about how, where, and what to render depending on the target device.

1.4.3.2 User Needs

- a.) ~~a.)~~ Users will want to configure displays and logic in an ~~configuration~~ environment that ~~depict~~ mimics what the graphic will “look like” on the target device.
- decisions can be made regarding real estate and scaling for multiple form factors
 - allows users to “picture” what the display will look like across devices
 - requires emulation of display devices from a view port point-of-view (displays may be forced into panels of appropriate size during test)
- ~~so that decisions can be made regarding real estate and scaling.~~

~~While designing displays users will want to see what that particular display will look like and how it will react on the various display targets for which it is intended. This will mean emulating the display and its logic on the development machine in a “simulate” mode for viewing.~~

- b.) If displays are intended for multiple display devices, it may be necessary for users to define “styles” for the display, one for each type of device. If no extra configuration is performed, displays will automatically take on “best fit” rules, but they may not be suitable in some instances, so configuration input will be required.
- allows certain components to be hidden on some devices to save space
 - allows for components to be re-located at runtime on some devices
 - allows for user to explicitly control how scaling takes place across form factors

It should be possible for users to “configure” a display (same components, widgets, logic, etc...) for multiple devices, and to “save” each configuration as one graphic but with multiple views or layouts (in some cases, some of the graphical elements will be invisible in certain configurations but not in others).

- c.) The user should not be burdened with understanding the technical mechanism taking place behind the scenes (whether it be stylesheets, javascript, or display server code) to make this one-graphic-multi-display principal “magic” happen take place. They will want to know that a single, yet, logical, display graphic entity exists somewhere in the system and appears to run seamlessly across multiple devices.

1.4.4 Dynamic Behaviors, Interactions, & Animations

1.4.4.1 Concept

Dynamic behaviors are the basic properties exposed and actions taken by the various components within a display graphic. These behaviors provide the mechanism(s) by which displays take on the dynamic look and feel based on how these behaviors have been configured and driven by the underlying display logic.

Interactions are the display logic driven connections between the various components on a display that coordinate and synchronize behaviors. For example, the user clicking a button and having that event change the visibility of another component on the same graphic. Typically, these interactions will be based on some combination of dynamic behavior properties; however, they could just as well be based on underlying process dynamic events (e.g. pump being turned on).

Animations are dynamic updates to components that are natively supported by the SVG specification. These behaviors tend to give components the capability to change in a visually rich, yet substantively thin way. In other words, animations can be used to “grab” the attention of the operator (flashing, spinning, bouncing, etc...), but do very little to actually change the state of the underlying information. Again, almost all of the stock properties of a display component can be animated, without the use of special display logic, since SVG supports it natively.

1.4.4.2 User Needs

- a.) Users will typically expect the behaviors of individual components to follow similar configuration requirements as do other components throughout the Starburn system (e.g. SFCs, Function Blocks, Equipment Modules, etc..). This means that, in general, configuration of the behaviors should be capable of being accomplished via dialog-based properties and browse-for-path type operations.
- b.) Users should not have to understand, unless they wish to view and edit the “raw ” code for the display, how any of the display logic is broken down between SVG-based animations vs. JavaScript functions. Regardless of the configuration mechanism for display interactions (either with the underlying control system or with other components), users should not be presented with choices on how to provide this functionality¹.
- c.) The following is a non-exhaustive list of some the more commonly used dynamic behaviors:
 - Text
 - Visibility
 - Scale
 - Position
 - Rotation
 - Skew

¹Essentially, there will likely be at least two ways in which to provide certain dynamic behaviors, either through native SVG support or by including additional JavaScript. The point is that the language and logic subsystem should make this determination for the user, and the user will never know or care which was selected.

- Enabled, Disabled
- Stroke (Pen Size)
- Fill Color
- Stroke Color
- Gradient (both linear and radial)
- Font
- Font Size
- Filter
- Images
-
- etc....

1.4.5 Behavior Classes or- Scriptlets

1.4.5.1 Concept

Part of the idea behind the Starburn development in general is to leverage object oriented concepts as much as possible, particularly from the stand -point of engineering configuration and productivity. Along these lines, the ideas of class-based displays and display components will be made available to end users both for direct re-use or for extension. The display language and logic subsystem will also take full advantage of code re-use and class-based “components” by utilizing libraries of both SVG and JavaScript procedures. Toward this end, the idea of behavior classes will be explored.

Behavior classes can be thought of as an encapsulation of the logic required to produce one or more of the dynamic behaviors discussed in sSection 1.24.4. -andThey will allow users to configure interactive display components to take on a specific behavior by “assigning” one or moreone such behavior classes to one or moreto those -components. A detailed use case scenario is included in sSection [2.2.2TBD] of this document [TBD], but a brief description is provided as follows. Suppose a user wishes to have several a particular widgets on a display be invisible upon the initial loading of the graphic. Obviously, an advanced user could easily delve into the underlying SVG and simply setup the appropriate SVG -animation or JavaScript code to accomplish this rather simple task. However, instead he/she could also select the components of interest, right-click and select the option to “Add Behavior”². This would then present a list of available behavior classes, possibly with some hints or clues as to what each does so selection is as easy as possible¹. In this case, the user would then selectchoose the behavior class called -hiddenOnLoad, which would then automatically perform the necessary steps within the display logic to make sure that components isare not shown when the graphic loads.

There will be potentially a large number of behavior classes, each providing some self-contained functionality, with possibly multiple source and multiple target components on which the script acts. These behavior classes will be “classes” in the sense that they will act on components found in display classes, and will resolve to the correct instance components at runtime as expected.

¹ The naming of behavior classes should make it fairly self-evident by itself, but tool tips and good context sensitive help will also be required.

1.4.5.2 User Needs

- a.) Users will want to easily recognize and understand, or at least have pretty good guess at, what the intended functionality of a behavior class is intended to do. A clear, precise naming scheme will be important as will adequate tool tips and good context sensitive help.
- b.) There are likely a fairly large, yet standard number of dynamic behaviors, interactions, and animations that process graphic designers typically utilize when putting displays together. The logic and support for a high percentage of these behaviors should be “canned” and supplied out-of-the box in the default library of behavior classes.
- c.) Users will want the ability to customize the default behavior classes by viewing and modifying the resulting logic, and, having done so, will want the ability to back-import these new behavior classes so they can be utilized in other places.

1.4.6 Display Logic Configuration

1.4.6.1 Smart Dialog and Property-Based Configuration

1.4.6.1.1 Concept

In general, most end users will not want to perform large amounts of customization via script-based programming. These users will be more comfortable and efficient by leveraging a more typical dialog/property-based configuration of their display logic. One way to accomplish this is to provide tab-like dialog helpers to configure the dynamic behaviors behind the display components. These dialogs will function in a “wizard” type mode¹ and allow for familiar browse and select- type configuration of component properties and process control system parameters.

This configuration option will also leverage the ability to associate **process dynamics** s with display components. Process dynamics are used to associate a component’s value or specific property with a real I/O point within the process control system runtime. For example, a static ~~<text>~~TEXT field might have a module’s OUT.CV associated with it, and therefore will automatically show the current value whenever the display is loaded. For a detailed description and scenario of process dynamics see the document ~~Starburn~~–Process Graphics Edit, Save, Download, Run General Scenarios.

Finally, using the behavior class concept discussed in Section 1.24.5, users will be able to graphically configure dynamic interactions and animations behind the components on a given display. Along with the behavior classes themselves, users will be able to configure some animations simply by using smart-dialog helpers to guide them through a series of configuration steps, which establish the desired effect. For example, the user would select a button and then ~~start by bringing up a helper~~launch a helper–dialog, which presents a list of events supported by the button. The user would select an event, say the “**onclick**” event, and then browse to a target component and select its visible property. Having done this, he/she would have associated the target component’s visibility with the source button’s clicked

¹ Not so much from a wizard look-and-feel, i.e. the Next>> and <<Back type guide, but more from the standpoint for helping the user navigate toward the desired result by taking into account information already known about the configuration (graying out superfluous options, presenting filtered lists, etc...).

event (some other configuration options may be presented to further customize the interaction, e.g. to set if the visibility toggles, etc...).

~~Configuration of display logic requires two distinct approaches, each designed to address the two extremes of user requirements. In general, most end-users will not want to perform large amounts of customization via script-based programming. These users will be more comfortable and efficient by leveraging a more typical dialog/property-based configuration of their display logic. On the other hand, there will undoubtedly be times and circumstances under which the standard, out-of-the-box behaviors will not be adequate. In these cases, the user(s) will want to leverage the underlying display logic script to customize and extend the default behaviors to meet special needs.~~

1.4.6.1.2 User Needs: Dialog-Based Configuration

Each component within a display must be able to enumerate its own specific properties and/or behaviors, along with any available values so that during configuration, users can easily configure the component(s) by browse and select type operations.

- a) Users of this type will want to be able to configure the component's dynamic behaviors (visibility, color, etc...) by selecting from a list of possible triggers (possibly a list of module parameters from the underlying control system) and setting which values signify what component behavior.
- b) Under more complex scenarios, users may require writing "expressions," not unlike done in other areas of Starburn configuration; ~~for example with SFC transition expressions.~~ Writing these behavior expressions should require as little code writing as possible by providing expression builders and browsing to other display components or control system parameters.
- c) Users will want to assign behaviors to groups of components all triggering off the same signal. An example might be multi-selecting a group of components on a display and assigning to them the "~~HiddenOnLoad~~hiddenOnLoad"² behavior class (doing so would automatically generate the behind the scenes script to set each component's visible behavior to false when the display loads).
- d) Users will expect dialogs to take into account as much knowledge about the situation as possible and to present choices, which make the most sense in that light. However, if wanted users will need the ability to override this intelli-sense, and to configure what might seem an unlikely scenario (for example select a button and circle, and have the circle be the "clicked" object and the button be the "target").

Users will want to assign components or groups of components to various layers within a display (for concepts on layers see the Starburn Runtime Operator Workspace Concept document). Assigning a component to a particular layer will automatically associate the component's behavior and logic (e.g. data gathering script or visibility) with that layer's state.

Users will need to associate various components with runtime or possibly historical data. For example, a slider bar component might be associated directly to an analog output. This should be accomplished by browsing to the module and having the various component properties (max., min., etc..) configured automatically.

1.4.6.2 Function Block-Based Logic Configuration

1.4.6.2.1 Concept

Many of the other configuration and engineering tools found within Starburn utilize the idea of function blocks to configure logic. In this situation, the “logic” is encapsulated in a graphical block, which performs a very specific and self-contained task (very similar to the aforementioned behavior class). Each block can then be linked to other blocks via a series of connecting lines, which themselves serve as inputs or outputs to the other logic blocks. A detailed scenario is included in Section T??.?[TDB] of this document [TBD], but essentially, this concept allows users to again graphically configure the dynamic behavior and interactions of the components found in a display. Like the idea of behavior classes, in the end, the display subsystem ultimately translates these function blocks into JavaScript code that becomes part of the overall display at runtime.

1.4.6.2.2 User Needs

- a.) Users will want a rich and fairly complete library of function blocks available that they can use to configure a wide-variety of dynamic behaviors. These blocks should be well named and provide tool tips to help users as they configure their displays.
- b.) Users will want the ability to group a set of blocks together and the import this “composite” back into the library to be used again in later situations.
- c.) Users may want to customize a single function block’s behavior by modifying the underlying script logic and then save and import the new block back into the library for re-use.

1.4.6.3 Script-Based Logic Configuration

1.4.6.3.1 Concept

There will inevitably be as small set of users that for any number of reasons either want or require the ability to directly edit and modify the logic of a display directly. These users will actually open and edit both the SVG and JavaScript behind a display and then test and debug the result.

1.4.6.3.2 User Needs: Script-Based Configuration

Advanced users, or those more familiar with programming/scripting constructs may want or need to configure display logic and behavior in a more powerful and direct way than possible through dialog-based or function block-based configuration. For this reason, all component dynamic behaviors must be actually governed by a behind-the-

~~scenes scripting language¹. These users will expect to develop in an environment very similar to today's advanced IDEs, like Visual Studio .Net.~~

- a) ~~To facilitate this both the non-programmer and script-savvy users,~~ displays will require at least two distinct “views”. ~~The first is—a a “form” view, which shows the user the current graphical layout of the formdisplay, and a. The second is a “code” view, which allows the user to edit and view the script behind the graphical components.~~
- b) Users will not be allowed to break any of the overall design constraints of the Starburn graphics, which means the editing environment cannot allow direct access to runtime APIs or other constructs that would break the one-display-many-devices concept.

Users will most likely want to maintain some type of versioning and control over the script behind the display, and this will need to be taken care of using VCAT or the like.

- c) Users will expect editing tools that increase productivity—Some-type of syntax checking, well-named auto-generated code, code navigation tools (find, goto, replace, etc..), intelli-sense and auto-complete functionality which lists the various display components and individual properties/behaviors will be required to ease codin color coding, etc..g.

Users will want to easily determine what code belongs to which component(s), so well named auto-generated functions and event handlers will be important.

Users will want to leverage other Starburn APIs from within the display logic (e.g. process history, alarms and events, advanced control, etc..), so the scripting environment should support this type of integration.

In some cases, users may want to integrate 3rd party based information by using 3rd party APIs, and thus the scripting environment should facilitate this to some degree².

1.4.7 Inter and Intra Inter and Intra Display Component Communications

1.4.7.1 Display Variables and Contexts

1.4.7.1.1 Concept

¹ ~~Most likely this language will either be JScript or C#, but others maybe considered. It also brings up the issue of what is the execution environment of this script.~~

² This opens some fairly complex issues regarding security and performance so it may only be “supported” in limited ways.

This model illustrates the fact classes of DeltaV Expert

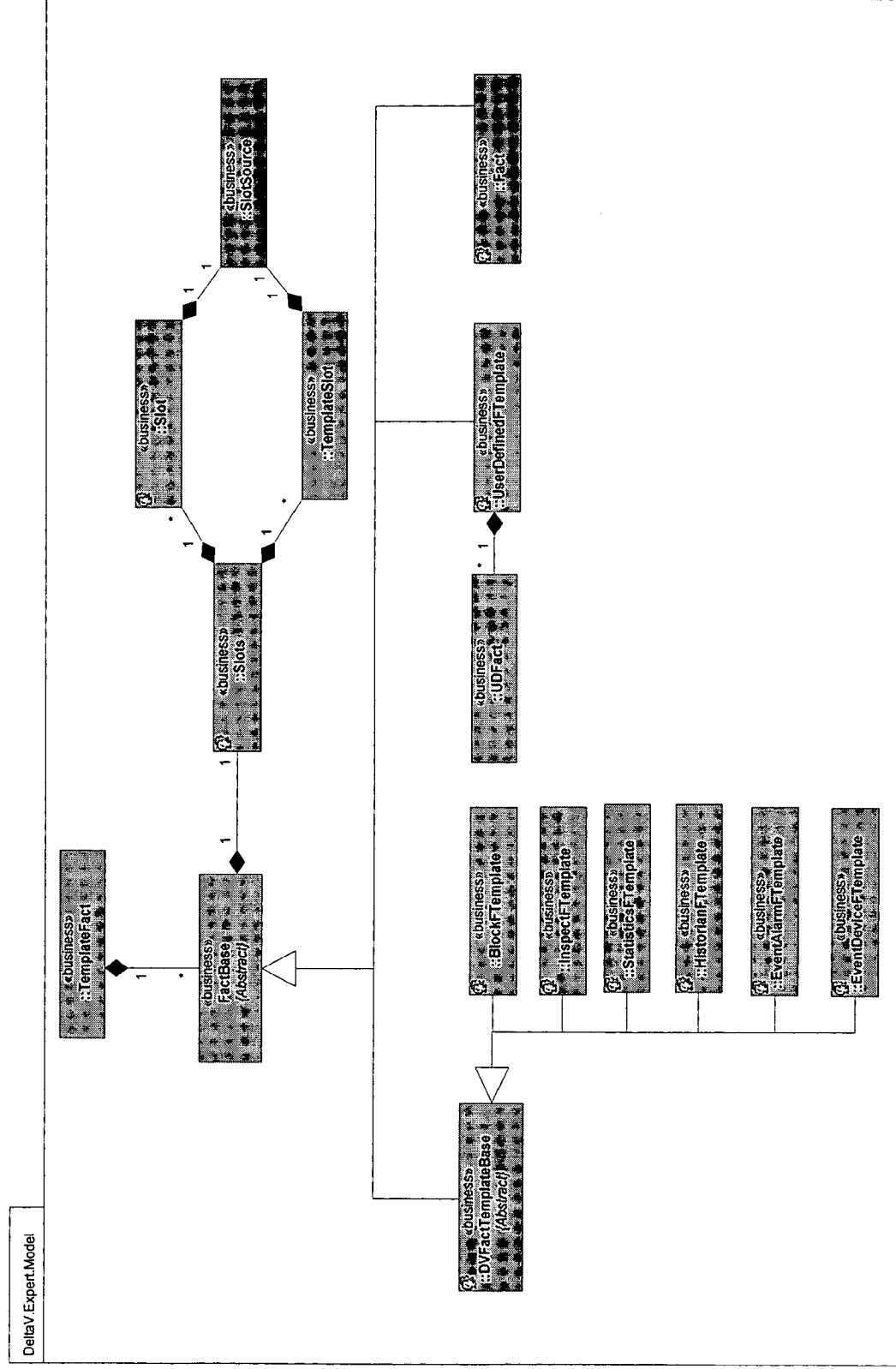


Figure 6 – Fact Class Diagram

This model illustrates the fact classes of DeltaV Expert

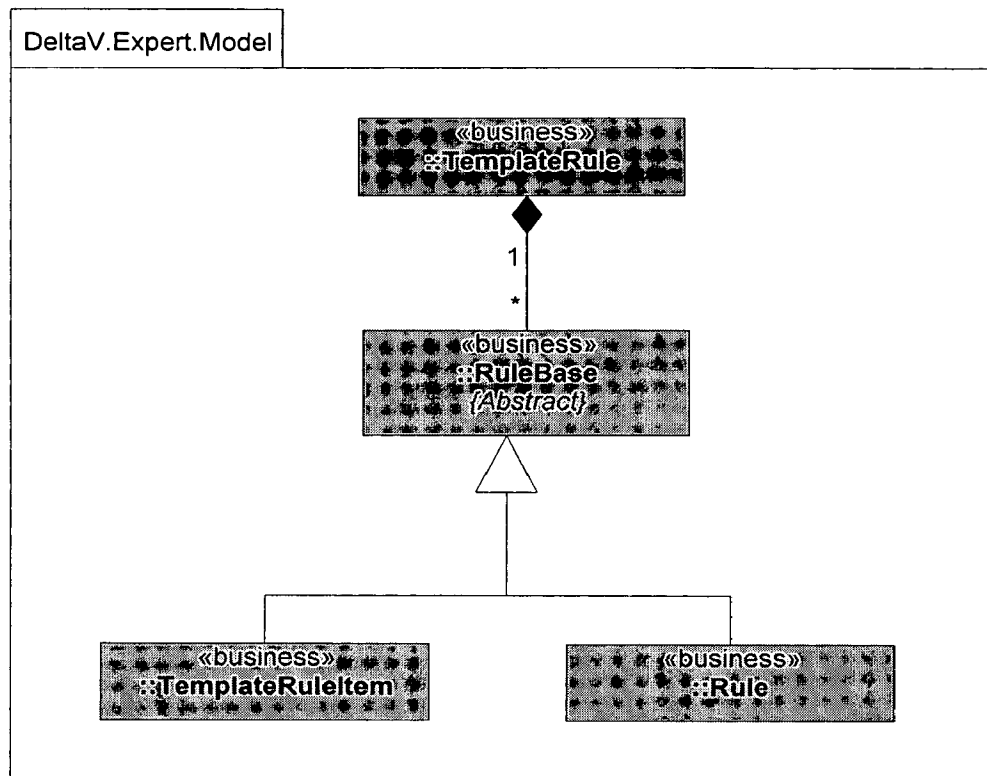


Figure 7 – Rule Class Diagram

4.2 Use Cases

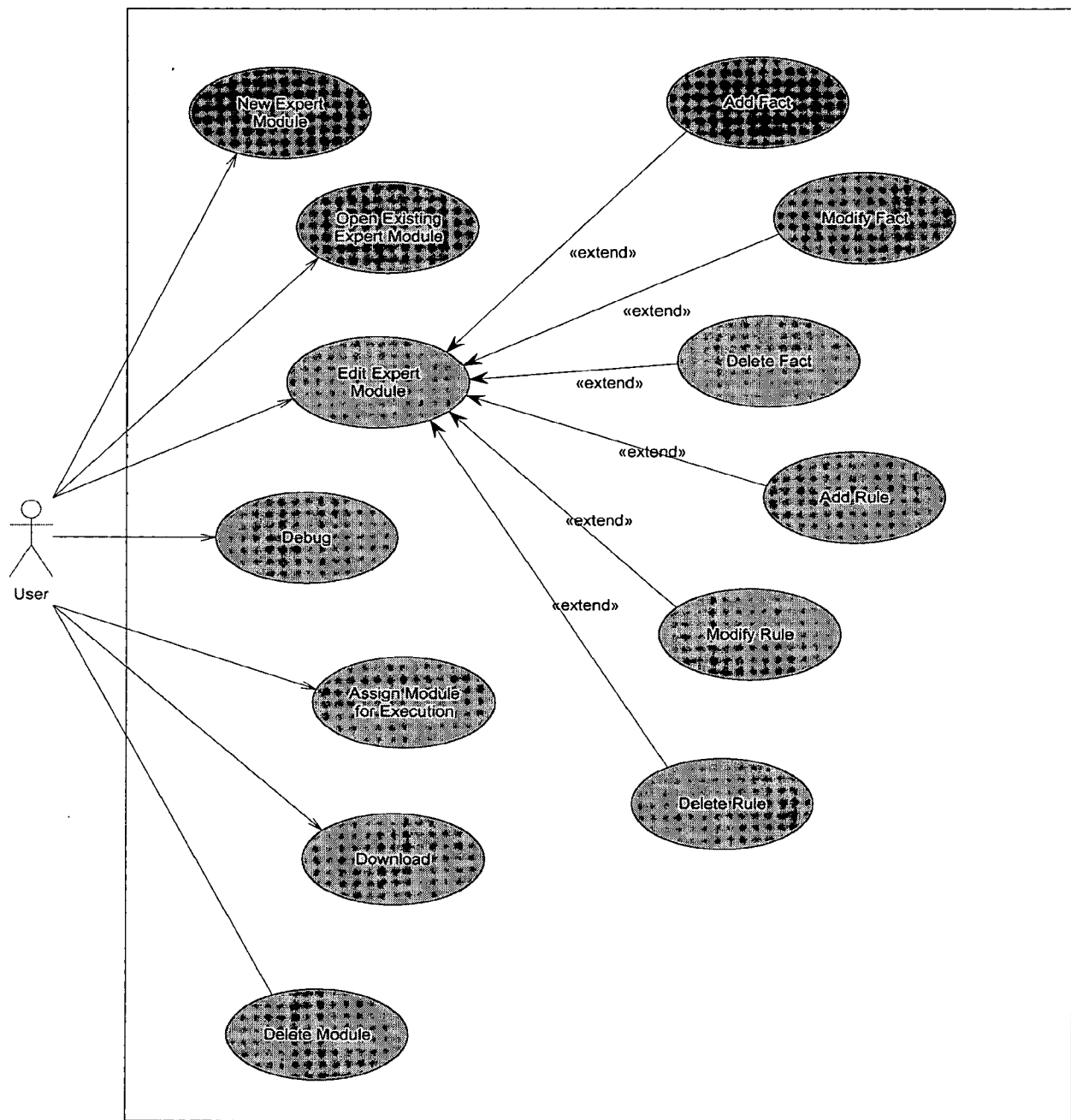


Figure 8 - Use Case Diagram

4.2.1 New expert module

4.2.1.1 Pre condition(s)

- a) Expert Studio is open

4.2.1.2 Post condition(s)

- a) A new expert module will be created and saved to the database.

4.2.1.3 User Actions/System Response

User Actions	System Response
User clicks "new" in Expert Studio	The application will create a blank DeltaV Expert Module

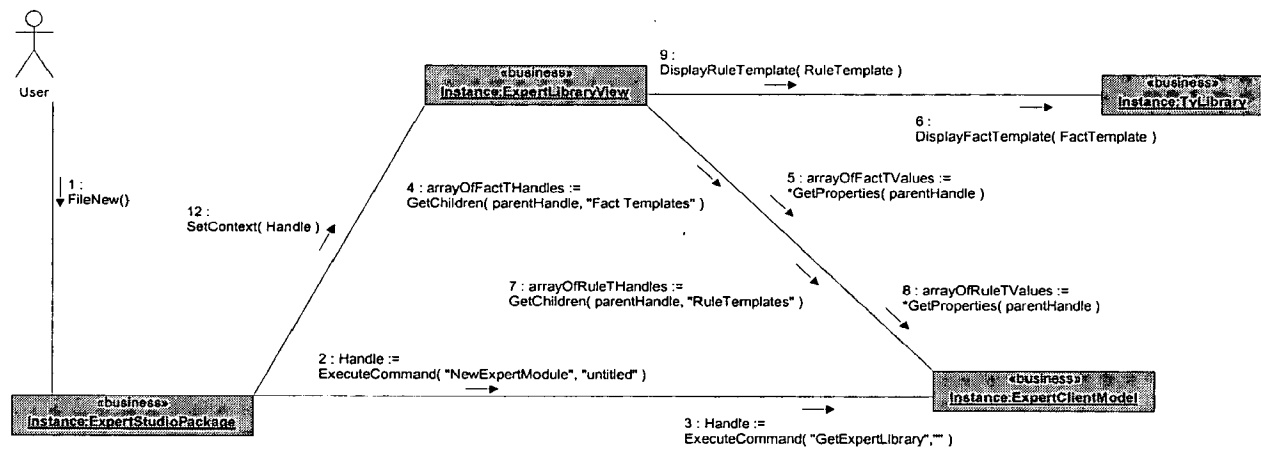


Figure 9 - Object collaboration diagram for creating a new expert module

4.2.2 Open existing expert module

4.2.2.1 Pre condition(s)

- a) At least one expert module has already been saved in the database.

4.2.2.2 Post condition(s)

- b) At least one expert module has already been saved in the database.

4.2.2.3 User Actions/System Response

User Actions	System Response
User clicks "open" in Expert Studio	A dialog box will appear giving a user the choice of what module to open based on a list of existing expert modules.
User selects a module and clicks "OK"	The application will attempt to load the user-specified expert module in Expert Studio.

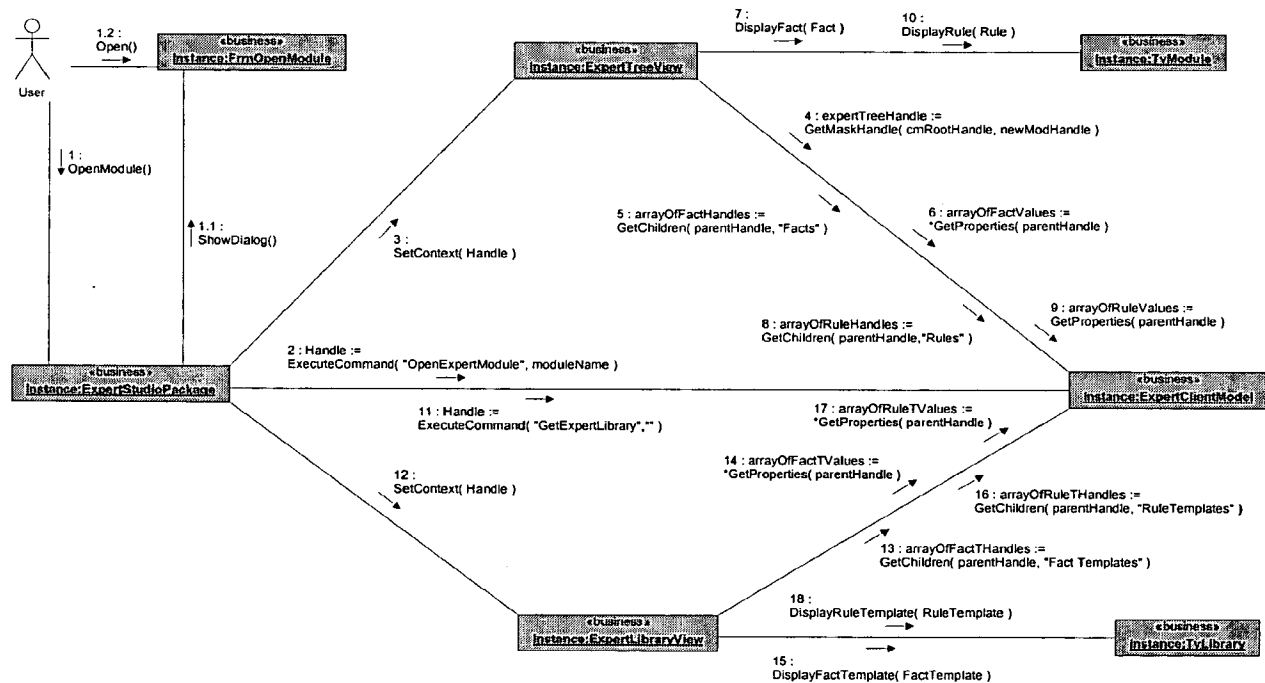


Figure 10 - Object collaboration diagram for opening an existing expert module

4.2.3 Edit expert module – Add fact

4.2.3.1 Pre condition(s)

- An existing expert module is opened in Expert Studio
- The expert module may not be assigned for execution

4.2.3.2 Post condition(s)

- All applicable changes to opened expert module will be saved in the database.

4.2.3.3 User Actions/System Response

User Actions	System Response
User drags a fact from the template view and drops it to the expert view	System will show a dialog displaying default and/or blank facts properties.
User fills-up fact properties	
User clicks on "Save" from the fact dialog	System creates, saves and assigns the fact to the working expert module

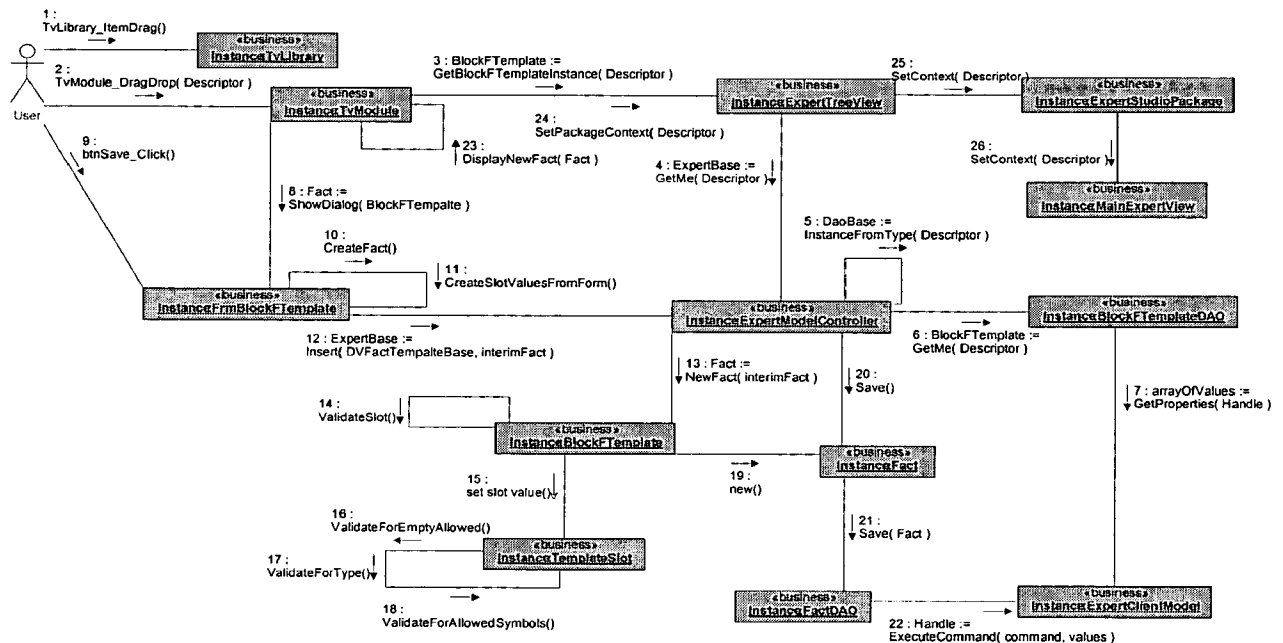


Figure 11 - Object collaboration diagram for creating a new fact in an expert module

4.2.4 Edit expert module – Modify fact

4.2.4.1 Pre condition(s)

- An existing expert module is opened in Expert Studio.
- There may be at least one existing fact
- The expert module may not be assigned for execution

4.2.4.2 Post condition(s)

- All applicable changes to the modified fact will be saved in the database.

4.2.4.3 User Actions/System Response

User Actions	System Response
User right clicks on an existing fact and chooses the "Properties" context menu	System will show a dialog displaying existing fact properties. User may use this dialog to view detailed fact properties or modify them.
User modifies fact properties	
User clicks on "Save" from the fact dialog	System saves user changes to the fact in the database.

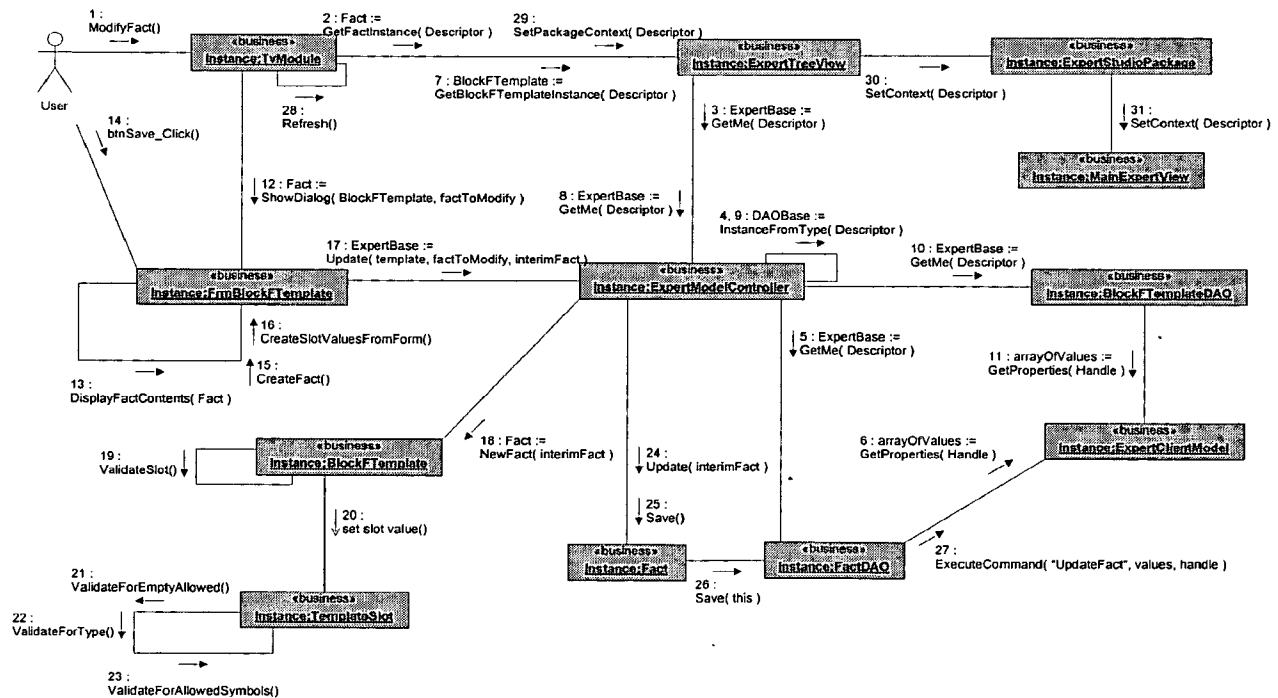


Figure 12 - Object collaboration diagram for modifying a fact in an expert module

4.2.5 Edit expert module – Delete fact

4.2.5.1 Pre condition(s)

- An existing expert module is opened in Expert Studio.
- There may be at least one existing fact
- The expert module may not be assigned for execution

4.2.5.2 Post condition(s)

- Fact will be deleted from the database.

4.2.5.3 User Actions/System Response

User Actions	System Response
User right clicks on an existing fact and chooses the "Delete" context menu	System will show a dialog asking the user for confirmation on the delete action.
User clicks on "OK" from the confirmation dialog	System removes the fact from the expert module and the database.

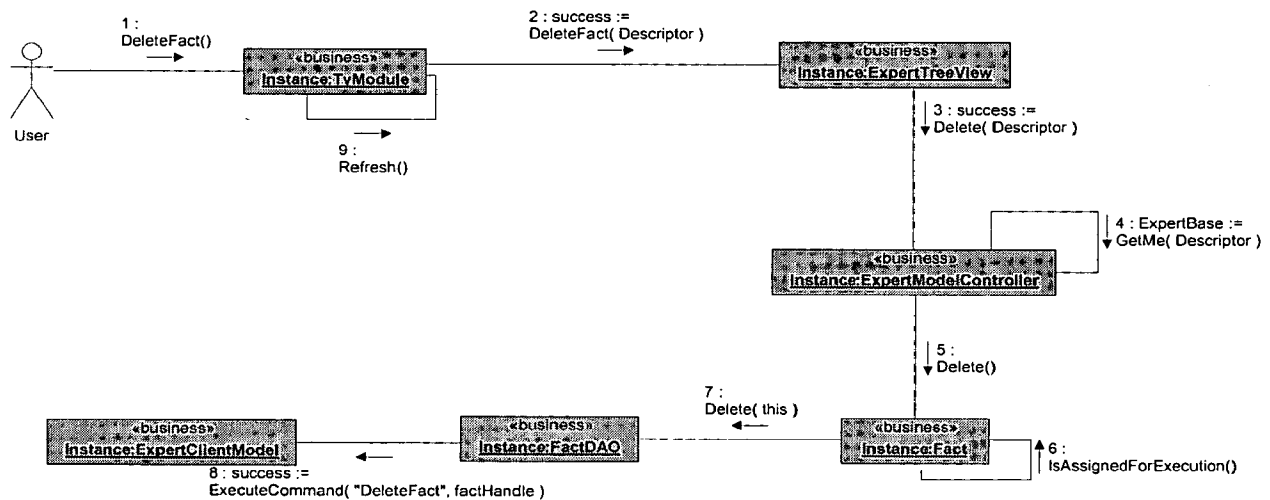


Figure 13 - Object collaboration diagram for deleting a fact in an expert module

4.2.6 Edit expert module – Add rule

4.2.6.1 Pre condition(s)

- c) An existing expert module is opened in Expert Studio

4.2.6.2 Post condition(s)

- b) All applicable changes to opened expert module will be saved in the database.

4.2.6.3 User Actions/System Response

User Actions	System Response
User drags a rule from the template view and drops it to the expert view	System will show a dialog displaying default and/or blank rule properties.
User fills-up rule properties	
User clicks on "Save" from the rule dialog	System creates, saves and assigns the rule to the working expert module

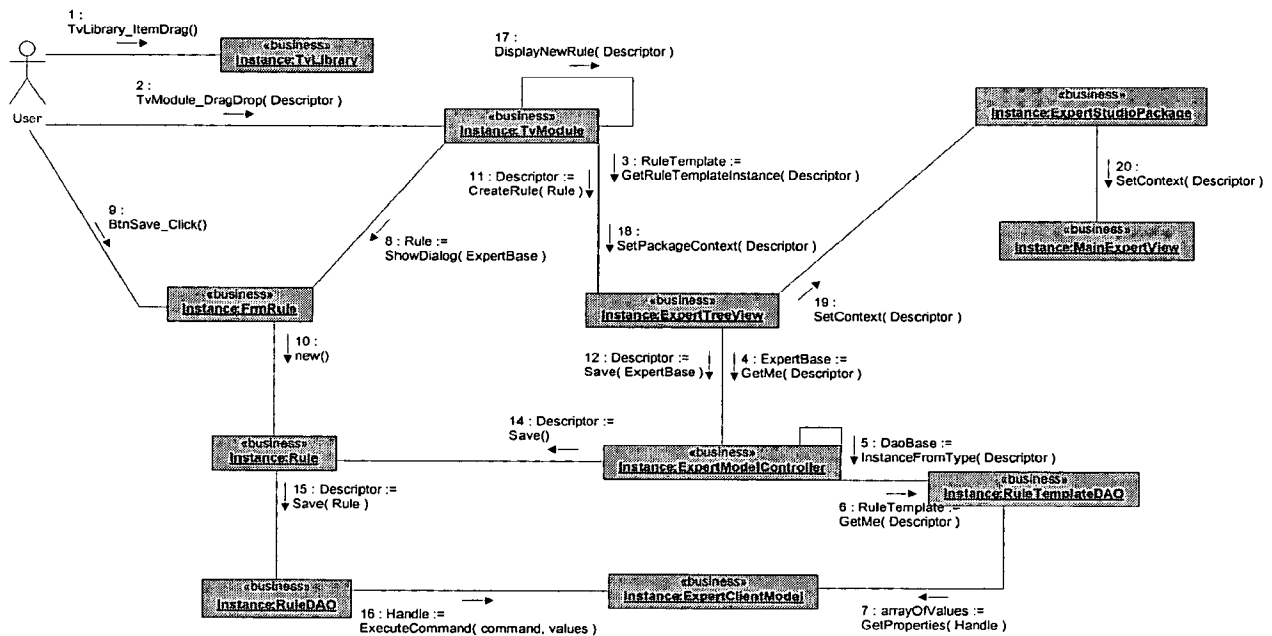


Figure 14 - Object collaboration diagram for creating a rule in an expert module

4.2.7 Edit expert module – Modify rule

4.2.7.1 Pre condition(s)

- d) An existing expert module is opened in Expert Studio.
- e) There may be at least one existing rule
- f) The expert module may not be assigned for execution

4.2.7.2 Post condition(s)

- b) All applicable changes to the modified rule will be saved in the database.

4.2.7.3 User Actions/System Response

User Actions	System Response
User right clicks on an existing rule and chooses the "Properties" context menu	System will show a dialog displaying existing rule properties. User may use this dialog to view detailed rule properties or modify them.
User modifies rule properties	
User clicks on "Save" from the rule dialog	System saves user changes to the rule in the database.

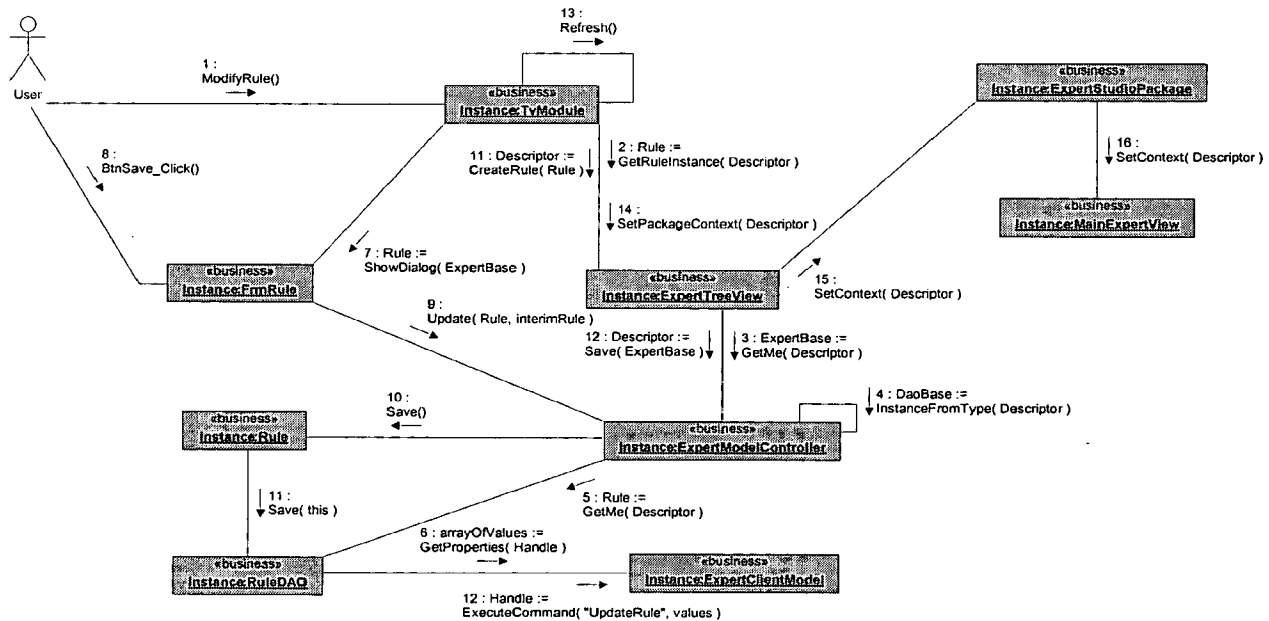


Figure 15 - Object collaboration diagram for modifying a rule in an expert module

4.2.8 Edit expert module – Delete rule

4.2.8.1 Pre condition(s)

- d) An existing expert module is opened in Expert Studio.
- e) There may be at least one existing rule
- f) The expert module may not be assigned for execution

4.2.8.2 Post condition(s)

- b) Rule will be deleted from the database.

4.2.8.3 User Actions/System Response

User Actions	System Response
User right clicks on an existing rule and chooses the "Delete" context menu	System will show a dialog asking the user for confirmation on the delete action.
User clicks on "OK" from the confirmation dialog	System removes the rule from the expert module and the database.

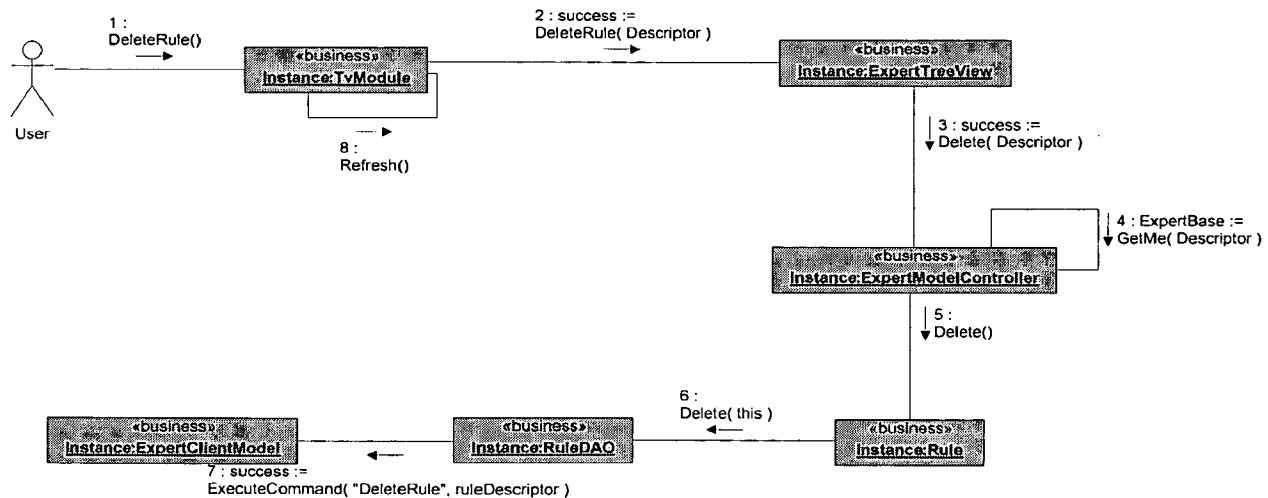


Figure 16 - Object collaboration diagram for deleting a rule in an expert module

4.2.9 Debug

4.2.9.1 Pre condition(s)

4.2.9.2 User Actions/System Response

User Actions	System Response

4.2.10 Assign module for execution

4.2.10.1 Pre condition(s)s

4.2.10.2 User Actions/System Response

User Actions	System Response

4.2.11 Delete module

4.2.11.1 Pre condition(s)s

4.2.11.2 User Actions/System Response

User Actions	System Response

4.2.12 Download

4.2.12.1 Pre condition(s)s

4.2.12.2 User Actions/System Response

User Actions	System Response

Process Simulation Configuration

Table of Contents:

1	INTRODUCTION	493
1.1	OVERVIEW	493
1.2	DEFINITIONS.....	493
1.3	REFERENCES	493
1.4	DELIVERABLES	493
2	OBJECTIVES	493
3	USER INTERFACES	494
4	OBJECT MODEL	495
4.1	DESIGN DIAGRAMS	495
4.2	USE CASES.....	499
4.3	CREATE PROCESS ELEMENT "TANK" USING PROCESS ELEMENT EDITOR.....	500
4.3.1	Pre Condition	500
4.3.2	Post Condition.....	500
4.3.3	Scenario	500
4.3.4	Collaboration Diagram	501
4.4	CREATE PROCESS MODULE "TANKMOD" USING PROCESS MODULE EDITOR.....	501
4.4.1	Preconditions	501
4.4.2	Postconditions.....	501
4.4.3	Scenario	501
4.4.4	Collaboration Diagram	502
4.5	CREATE PROCESS MODULE "TANKMOD" USING PROCESS MODULE EDITOR FROM A "TEMPLATE".....	505
4.6	VIEW ONLINE MODE USING PROCESS MODULE EDITOR.....	505

Table of Figures:

Figure 1 – Process Element Editor	494
Figure 2 – Process Module Editor	495
Figure 3 – Process Module Object Model	496
Figure 4 - Classes for the Simulation Configuration Editor	497
Figure 5 - Class Association for the Simulation Prototype	498
Figure 6 – Process Module Use Cases	499

1 Introduction

1.1 Overview

Process Simulation is a new feature in the DeltaV System that will allow plant operations to be simulated in DeltaV for both process optimization and operator training purposes. The process simulation package will have a configuration editor and an online mode. This document describes use cases for the configuration editors.

There are two editors needed to configure the process simulation. The process element and the process module editors.

a.) Process Element Editor

The Process Element Editor provides a way to create process elements which becomes the basic building blocks of a process module.

b.) Process Module Editor (Offline and Online mode)

The process module editor allows the composition of a process module by inserting and connecting process and elements to simulate process plant scenario.

1.2 Definitions

1.3 References

1.4 Deliverables

1.4.1 Process Element configuration editor

1.4.2 Process Module configuration editor.

2 Objectives

The objective for this prototype is to define and show user scenarios pertaining to simulation process configuration and to investigate plausible ways of interfacing with the Starburn Client Model, the Diagram Control, the Starburn Controls, and the Workspace.

3 User Interfaces

Configuration Workspace																																				
Menu																																				
Tool Bars																																				
<p><i>Tree View</i></p> <p>CONNECTIONS</p> <p>Steam</p> <p>Air</p> <p>Fuel</p> <p>Water</p> <p>Exhaust</p> <p>Create Graphic View</p>	<table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 25%; text-align: center; padding: 5px;">Initial</td> <td style="width: 25%; text-align: center; padding: 5px;">Run</td> <td style="width: 25%; text-align: center; padding: 5px;">Post</td> <td colspan="3" style="width: 25%; text-align: center; padding: 5px; background-color: #cccccc;">Input/Output/Path</td> </tr> </table> <div style="border: 1px solid black; padding: 10px; margin-top: 5px;"> <p>Number of Paths : <u> 2 </u></p> <p>Number of Inputs : <u> 3 </u></p> <p>Number of Outputs: <u> 2 </u></p> </div> <table border="1" style="width: 100%; border-collapse: collapse; margin-top: 5px;"> <tr> <td style="width: 15%; text-align: left; padding: 5px;">Name :</td> <td style="width: 15%; text-align: center; padding: 5px;">IN1 Steam</td> <td style="width: 15%; text-align: center; padding: 5px;">IN2 Air</td> <td style="width: 15%; text-align: center; padding: 5px;">IN3 Fuel</td> <td style="width: 15%; text-align: center; padding: 5px;">OUT1 Water</td> <td style="width: 15%; text-align: center; padding: 5px;">OUT2 Exhaust</td> </tr> <tr> <td style="text-align: left; padding: 5px;">Type :</td> <td colspan="5" style="border-top: 1px dashed black; padding: 5px;"></td> </tr> <tr> <td style="text-align: left; padding: 5px;">Path1:</td> <td style="text-align: center; padding: 5px;">X</td> <td></td> <td></td> <td style="text-align: center; padding: 5px;">X</td> <td></td> </tr> <tr> <td style="text-align: left; padding: 5px;">Path2:</td> <td></td> <td style="text-align: center; padding: 5px;">X</td> <td style="text-align: center; padding: 5px;">X</td> <td></td> <td style="text-align: center; padding: 5px;">X</td> </tr> </table>						Initial	Run	Post	Input/Output/Path			Name :	IN1 Steam	IN2 Air	IN3 Fuel	OUT1 Water	OUT2 Exhaust	Type :						Path1:	X			X		Path2:		X	X		X
Initial	Run	Post	Input/Output/Path																																	
Name :	IN1 Steam	IN2 Air	IN3 Fuel	OUT1 Water	OUT2 Exhaust																															
Type :																																				
Path1:	X			X																																
Path2:		X	X		X																															
<p><i>Parameter View</i></p> <p><i>Diameter</i></p> <p><i>Height</i></p>	<p><i>Alarms</i></p> <div style="border: 1px solid black; height: 40px; margin-top: 5px;"></div> <p style="text-align: right; margin-top: 10px;"><i>New alarms</i></p>																																			

Figure 1 – Process Element Editor

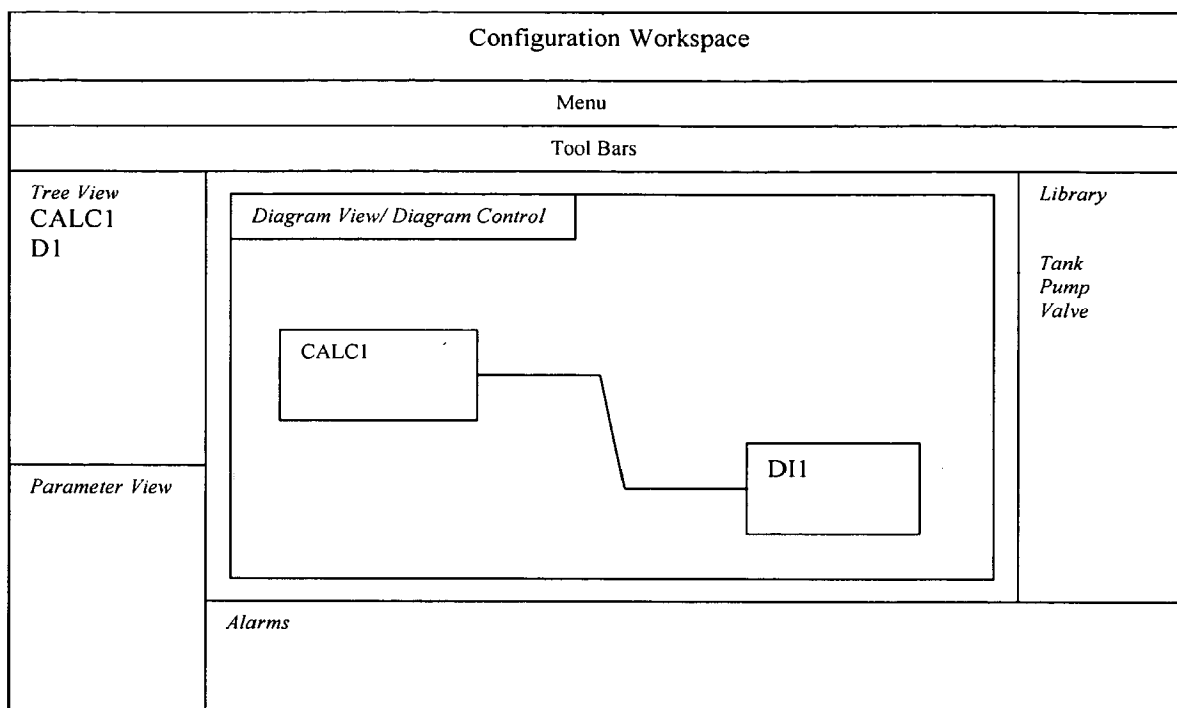


Figure 2 – Process Module Editor

4 Object Model

4.1 Design Diagrams

This object model illustrates how Process Module Objects are mapped into the existing model.

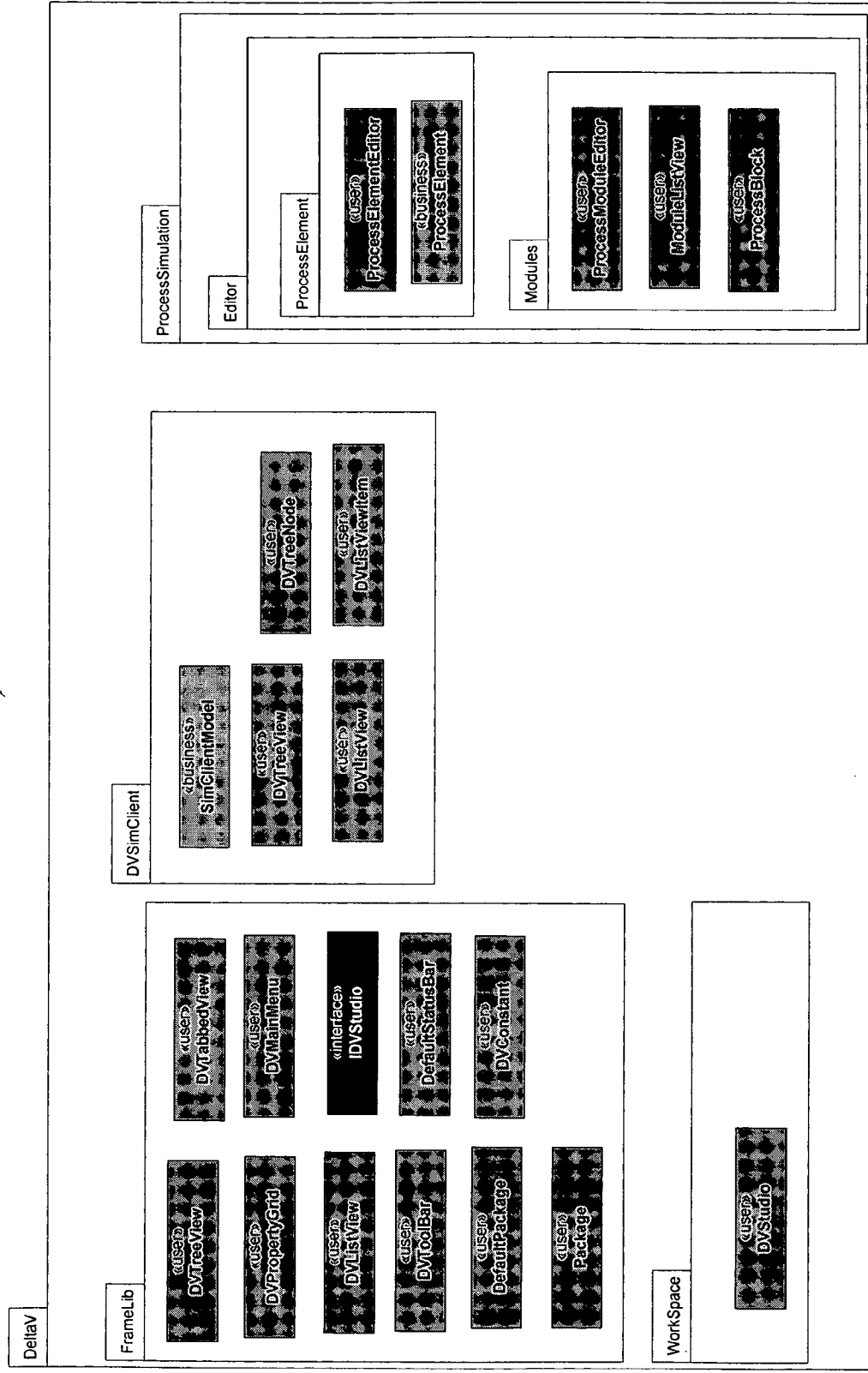


Figure 4 - Classes for the Simulation Configuration Editor

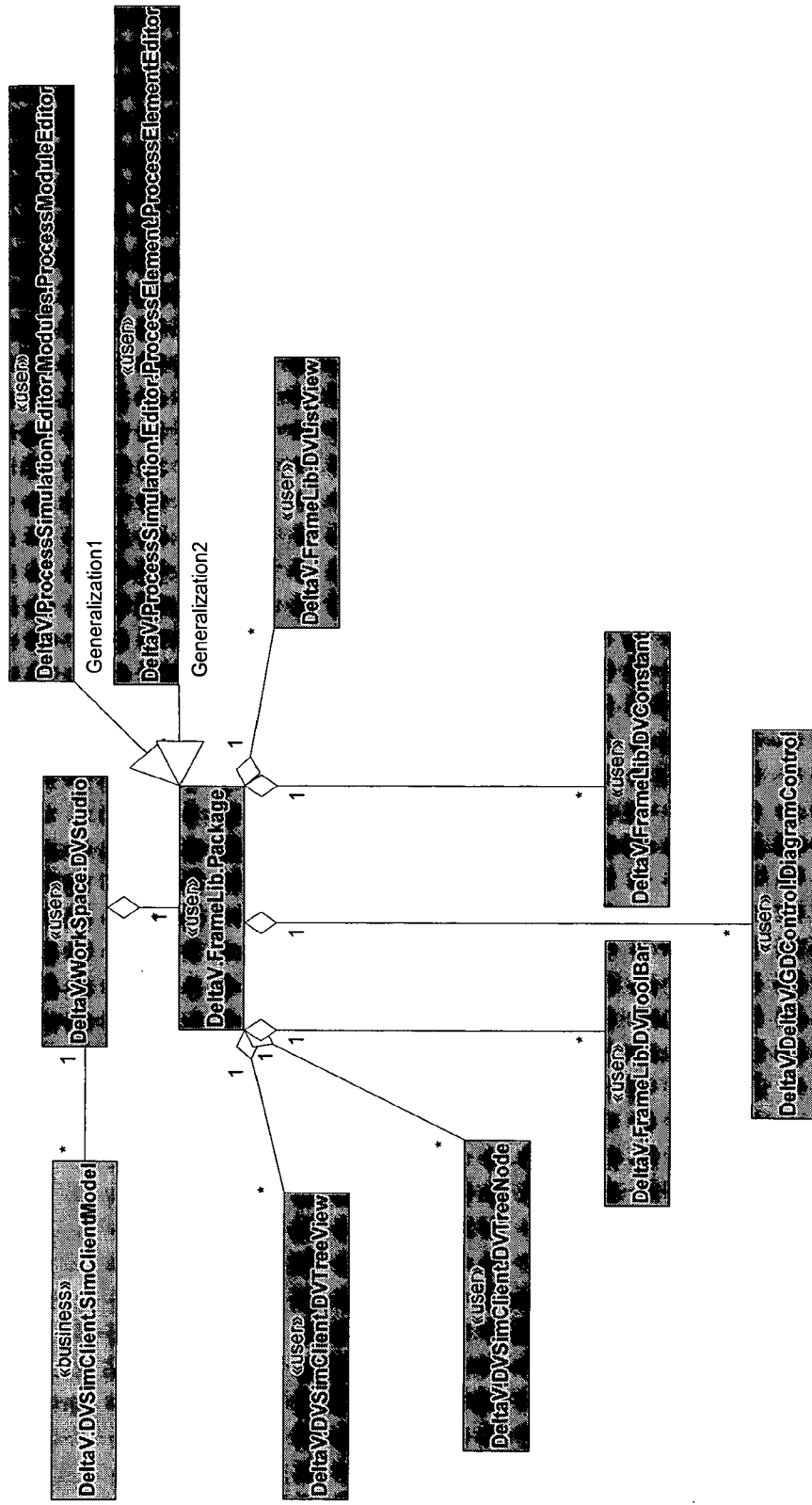


Figure 5 - Class Association for the Simulation Prototype

4.2 Use Cases

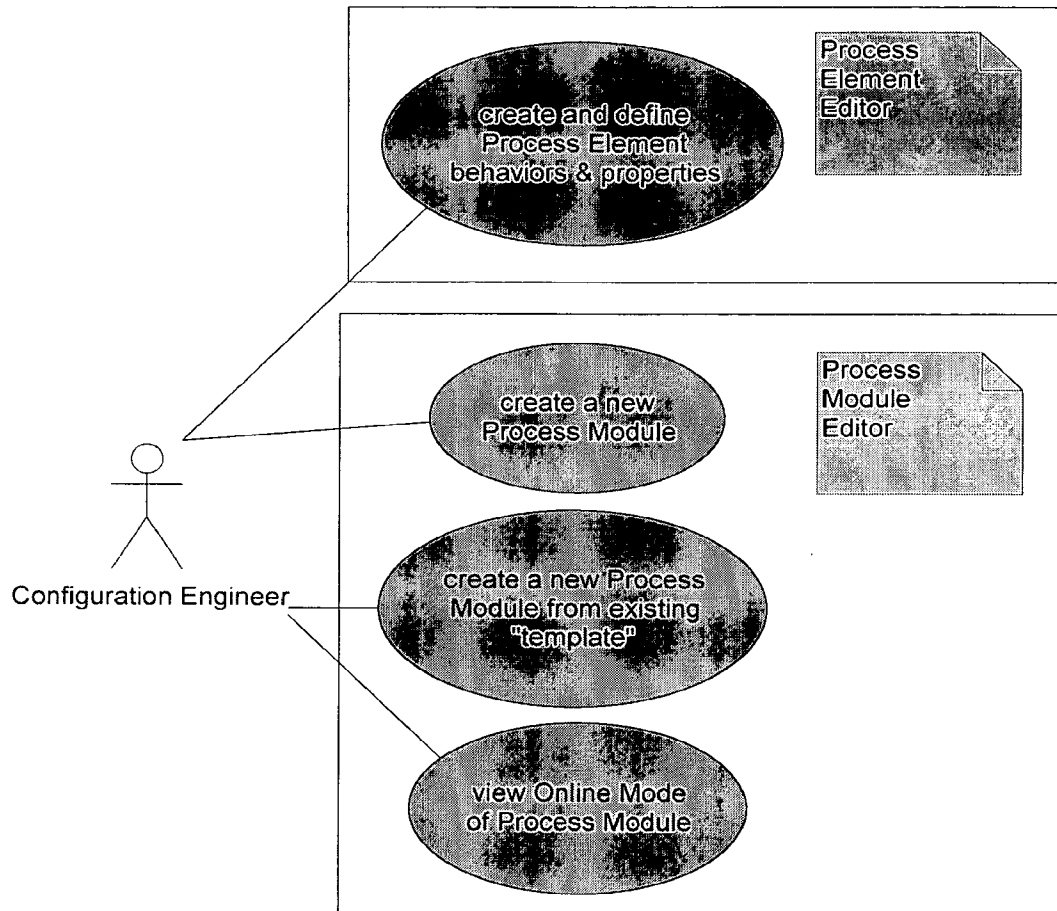


Figure 6 – Process Module Use Cases

4.3 Create Process Element “Tank” using Process Element Editor.

4.3.1 Pre Condition

Process Element “Tank” does not exist.

4.3.2 Post Condition

Tank created with algorithm.

Tank has properties diameter, height, number of inputs, number of outputs, alarms added.

4.3.3 Scenario

1. User launches Process Element Editor.
2. Process Element Editor loaded into Workspace as a Package.
3. Package loads “blank”.
4. User selects “New ”.
 - 4.1 Editor presents “New” dialog.
 - 4.2 User selects “Embedded Algorithm”.
 - 4.3 Editor selects process element “Untitled” with ST algorithm.
 - 4.4 Editor creates Tree View, Parameter View, and Tabbed Content with algorithm entries for “Initial”, “Run” “Post”, “IN” and “OUT”.
5. User defines Inputs/Outputs/Paths.
 - 5.1 User selects Input/Output/Path tab.
 - 5.2 User sets number of Inputs = 3, number of Outputs = 2, number of paths = 2.
 - 5.3 User sets name, type, etc, for each inputs and outputs.
 - 5.4 User defines path.
6. User defines Parameters.
 - 6.1 User selects Parameter View.
 - 6.2 User adds parameter “Diameter”.
 - 6.3 User adds parameter “Height”.
7. User defines Alarm.
8. User defines Initial algorithm.
 - 8.1 User selects “Initial” tab.
 - 8.2 Editor presents ST Editor.
 - 8.3 User enters ST.
9. User defines Run algorithm.
10. User defines Post algorithm.
11. User selects “Save” Process Element as “3_Input_Tank”.

4.3.4 Collaboration Diagram

4.4 Create Process Module “TankMod” using Process Module Editor

4.4.1 Preconditions

Process Module Package is Open with no current process module.

Mode is immediate update.

4.4.2 Postconditions

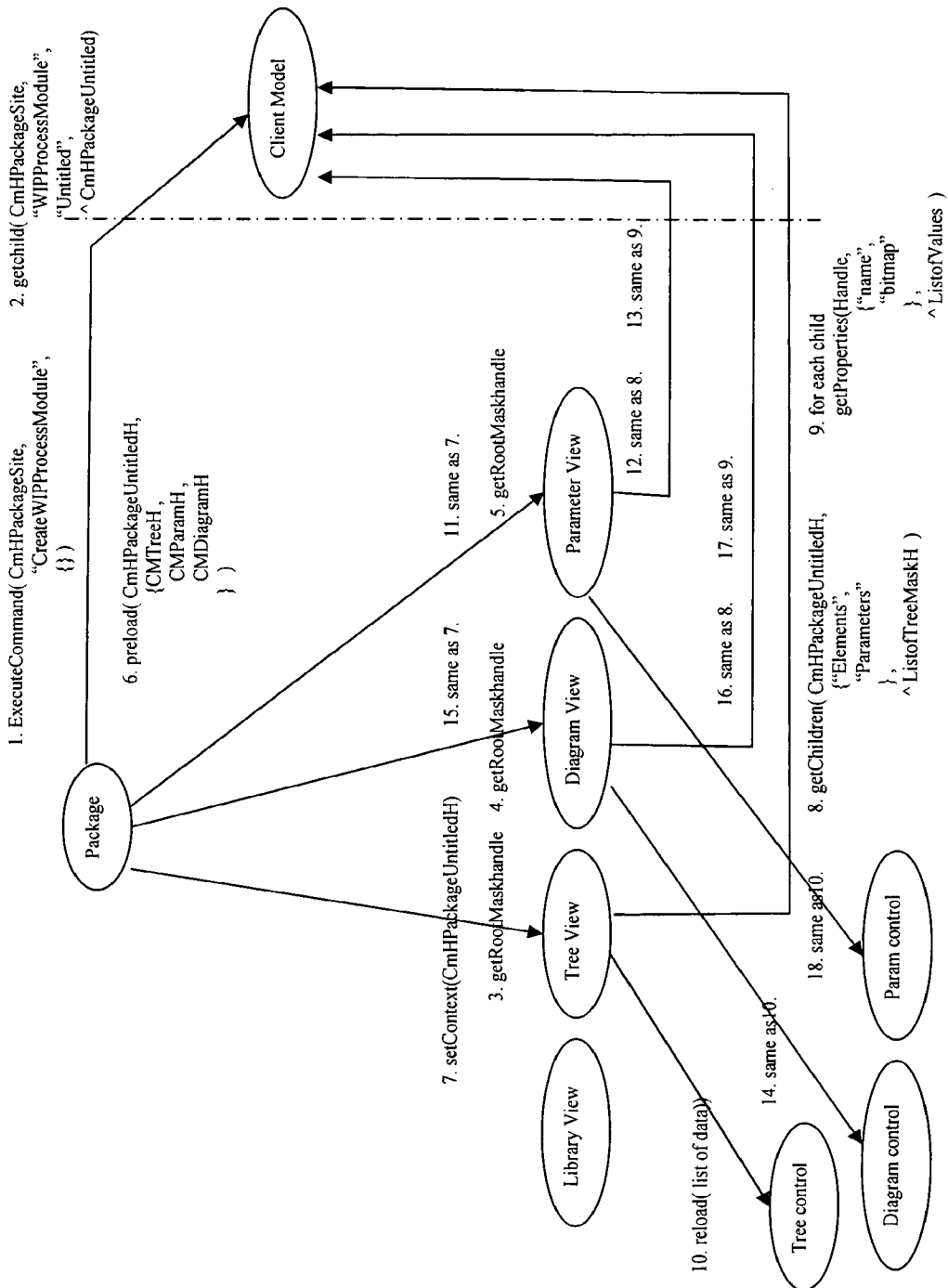
Process Module ‘TANKMOD’ created.

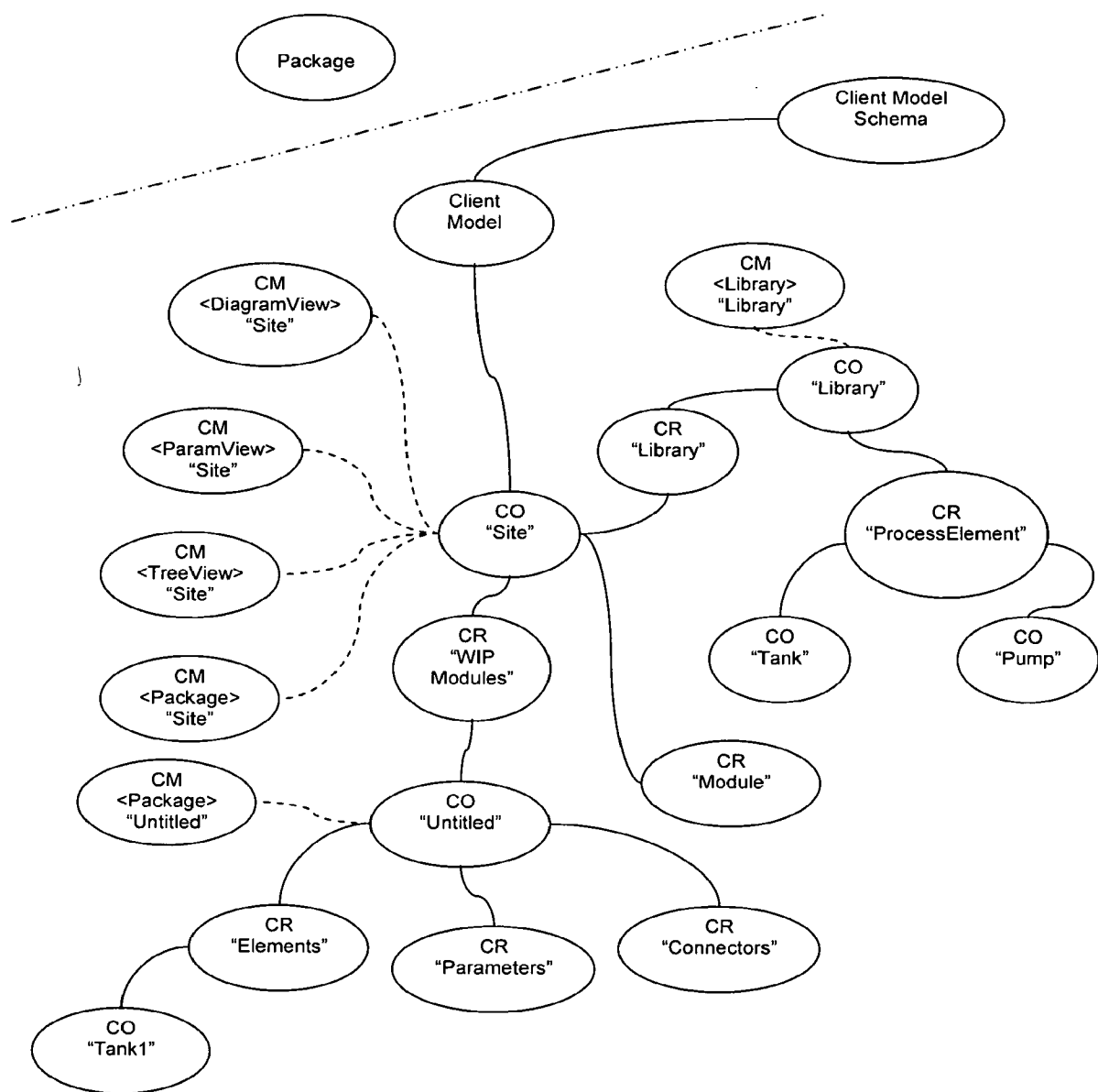
4.4.3 Scenario

1. User selects File – New.
Editor presents New Dialog.
2. User selects New.
Editor creates “Untitled” Process Module in DB & replicate.
3. User places elements on diagram.
 - Stream start
 - Pump
 - Valve
 - Tank
 - Stream end
4. User connects with wires – for each link.
 - 4.1 Select Connection tool.
 - 4.2 Click on start Connector.
 - 4.3 Click on end Connector.
5. User selects File-Save As with name “TankMod”

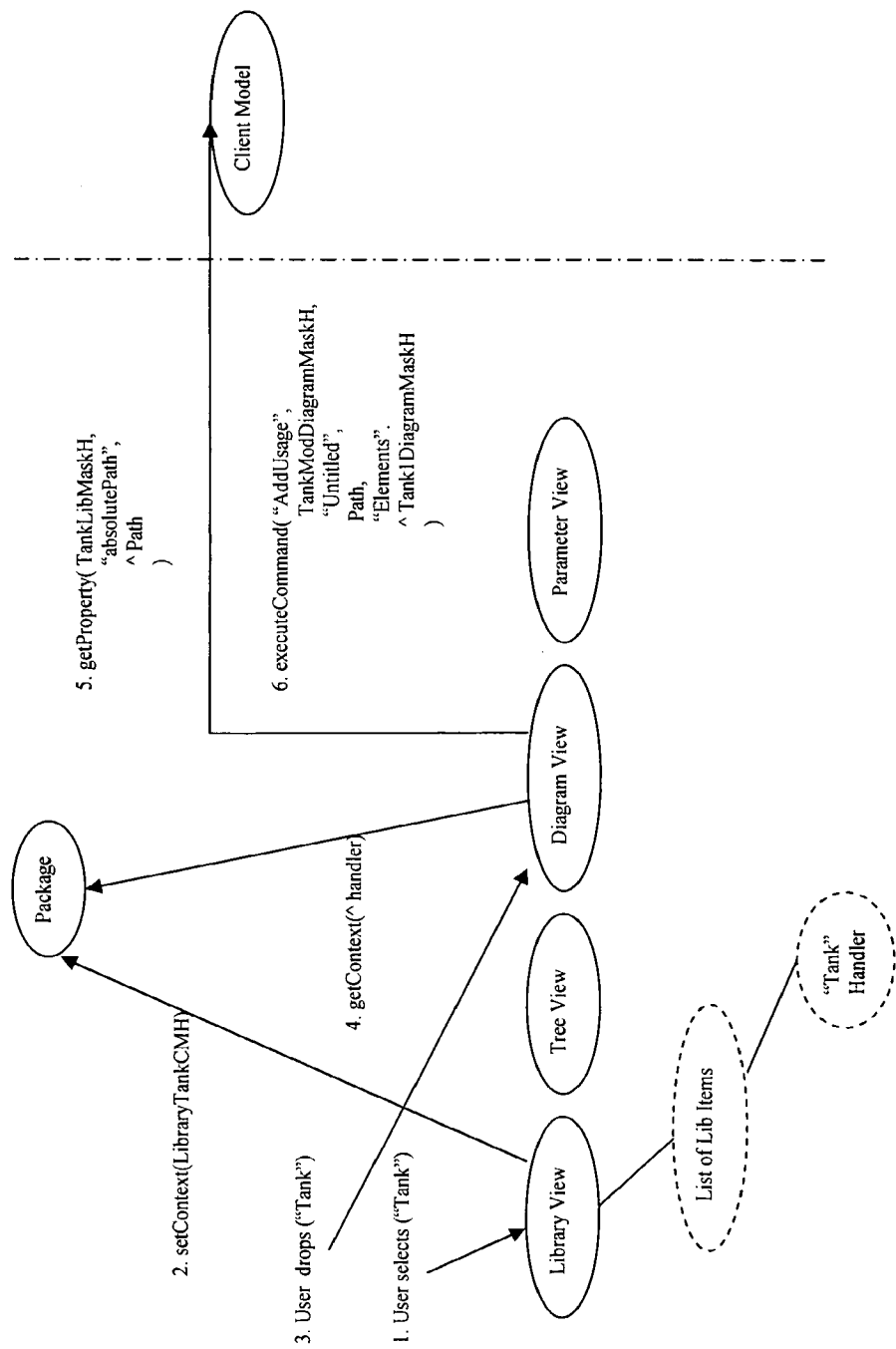
4.4.4 Collaboration Diagram

Scenario (2) User selects New.





Scenario (3) User places element on diagram. (“Tank” element dragged and dropped)



4.5 Create Process Module “TankMod” using Process Module Editor from a “Template”

4.6 View Online Mode using Process Module Editor

Process Modules
for
Improved Control, Abnormal Situation
Prevention, and Simulation

Table of Contents:

1	INTRODUCTION	510
1.1	OVERVIEW	510
1.2	OBJECTIVES.....	510
1.3	DEFINITIONS.....	510
1.4	REFERENCES	511
2	REQUIREMENTS.....	512
2.1	GENERAL	512
2.2	PROCESS GRAPHIC ELEMENTS	512
2.2.1	Process Connections	513
2.2.2	Process Actuators.....	515
2.2.3	Measurements and Property Elements	518
2.2.4	Processing Elements	520
2.3	PROCESS GRAPHIC STREAMS	527
2.3.1	Mass / Composition	527
2.3.2	Energy.....	527
2.4	LOCAL PANELS AND SUBSYSTEMS (INSTRUCTOR LAYER ONLY)	528
2.4.1	Standard Actuators	528
2.4.2	Subsystems	528
2.5	PROCESS MODULES / SIMULATIONS	529
2.5.1	Actuator Elements.....	529
2.5.2	Processing Elements	529
2.5.3	Process Measurement Elements.....	529
2.5.4	Connections	529
2.6	HIGH FIDELITY SIMULATIONS.....	530
2.6.1	General	530
2.6.2	Operator Training	531
2.7	EXAMPLE DISPLAYS.....	531
2.7.1	Boiler Example.....	531
2.7.2	Column Example.....	532
2.7.3	Custom Processing Example – Kamyir Digester.....	532
2.7.4	Evaporator Separator Example	533
2.7.5	Mixer Heater Example	533
2.7.6	Reactor Example	534
2.7.7	Spray Dryer Cyclone Separator Example.....	534
2.7.8	Tank Example.....	535
3	CONCEPT	536
3.1	OVERVIEW	536
3.1.1	Display Layers – Off-line Simulation	537
3.2	PROCESS GRAPHIC DISPLAY.....	538
3.2.1	Calculation and Control.....	538
3.2.2	Properties and Measurement	538
3.2.3	Actuator Elements.....	539
3.2.4	Processing Elements	539
3.3	PROCESS MODULE	540
3.3.1	Simulation Blocks.....	541
4	SCENARIO.....	546
4.1	OVERVIEW	546
4.2	PROCESS GRAPHIC AND PROCESS MODULE DESIGN.....	547
4.2.1	Process Graphic	547
4.2.2	Thumbnail View of the Process Graphic.....	548

4.2.3	Process Module	549
4.3	PROCESS SIMULATION USING HYSYS	550
4.3.1	Dynamic Simulation	550

Figure 7. Vinegar Production.....546

Figure 8. Vinegar Control Strategy.....547

Figure 9. Process Graphic Display.....548

Figure 10. Process Graphic Display Composite548

Figure 11. Overview Process Graphic Displays549

Figure 12. Process Module Display.....549

Figure 13. Acetic Acid High Fidelity Simulation550

Figure 14 High Fidelity Simulation Import/Export Mappings.....551

1 Introduction

1.1 Overview

Using Process Simulation for Operator Training, Plant Design, and IO Checkout is not a new idea. The current versions of DeltaV provide an interface through OPC with documented extensions to DeltaV to allow simulators such as Mimic and HYSYS to exchange parameters and control the simulation¹. The current technology, however, is not integrated in several important ways including configuration, alarming, and control.

Adding new capabilities to process graphics makes it possible to both add new behaviors and integrated customized behaviors with graphical items. These capabilities can be exposed “behind-the-scene” as Process Graphic Displays are built. These capabilities are exposed in what we refer to as “Process Modules”.

This concept document summarizes the requirements for the integration Process Graphics and Process Modules.

1.2 Objectives

The objectives for this project are:

1. Improve Control
2. Reduce plant shutdowns through Abnormal Situation Prevention
3. Reduce project engineering effort
4. Provide a framework for Operator Training
5. Improve integration with High Fidelity Simulation Applications

1.3 Definitions

Term	Definition
Graphics Primitives	These are built into the display configuration model along with their behavior. These include lines, polylines, ellipses, arcs, rectangles, text, images, composite (group). Each primitive in the editor has an SVG equivalent (although one editor primitive may be translated into multiple SVG objects, e.g. multi-line text, gradient filled objects etc.) User variables can be associated with any primitive. Variables are used by dynamic behaviors to control the algorithm and how items are displayed.
Graphics Group (composite)	Graphics primitives and other groups can be grouped together to form a grouped item which can behave as a single entity for editing purposes. User variables can be associated with any group. Variables are used by dynamic behaviors to control the algorithm and how items are displayed.
Component Primitives	Pre-built components that are built by EPM and distributed as components. Component Primitives have built-in behavior which can range from simple to complex. Component primitives have behavior which allows them to be configured. These components are for providing specialized interactions or controls. E.g. Trend controls, advance control interactions etc.
Process Graphic Item (aka - Library Item, either Template or	Are pre-built single primitives or groups, including any defined variables and dynamic behavior. We will provide libraries of composites for common control equipment such as PUMPS, VALVES etc. Users can also create their own library of composites. Library items can be treated as templates, in which case a copy of the item is made if it is placed on a display, or classes in which case a link to the library item is made if it is placed on a display. When an instance of a class is created on another display the

¹ Current Simulate packages allow the users to enable simulation, perform fast save/restores, etc.

Term	Definition
Class)	user can override the public parameters defined in the class, together with its viewport characteristics (position, size and rotation).
Display	A display is a graphics group which is capable of being displayed independently. It includes the definition of the "canvas" on which the display is drawn (its size, background color or background image). Public variables defined for a display can be used as parameters when the display is called up. For example, the color of pipework for a display could be controlled by a variable PIPEWORK_COLOR. When the display is called up the parameter PIPEWORK_COLOR could be set to blue.
Display Class	A display class is extremely similar to a display except that one or more of its public parameters are used as an alias in process value paths. For example, a display class could have process value paths including the alias MODULE (e.g. <code>##MODULE#/ATTR.FIELD</code>). MODULE itself is a parameter of the display. When the display is called up the parameter MODULE could be set to FIC-101A.
Display Template	Is exactly the same as a display except that it is placed in the library rather than associated with a particular equipment item, module, or item of hardware. A display template can be used as a starting point for creating displays.
Process Objects	Library items can be tagged. Tagged graphical items can be referenced by other configuration items as well as the runtime. Smart Process Objects are normally equipment items such Valves, Pumps, Tanks, Heaters, Mixers, etc.
Process Links (Streams)	Smart Process Links are also referred to Streams in other simulation and optimization packages. Smart Process Links can have compositions associated with them. When they compositions associated with them the compositions flow along the stream through the Smart Process Objects. Smart Process links can also have calculations associated with them. This allows other them to be referenced by other configuration items as well as the runtime. They are most often associated with Pipes.
Process Algorithm	Similar to existing algorithm types, Process Algorithms are defined for generalized use. Each process graphic may have a process module associated with and a process algorithm defined for it. Algorithm types include Mass Balance, Energy Balance, Simple Composition, Routing, Efficiency, Optimization, Economic Calculations, as well as custom algorithm types.
Process Module	Can be defined for Process Graphics. They have special properties including, Mode, Status, and Alarm Behavior. Can be assigned to workstations. Execute process flow algorithms.
Smart Process Database	Stores information about Smart Objects, Links, Algorithms, and Modules. Similar to Component and OO Design tools, the database will allow the users to define multiple views against the same database items (e.g. class diagrams, state diagrams, collaboration diagrams, etc.)

1.4 References

2 Requirements

2.1 General

1. **Improved Control** – Provide the following calculations
 - a. Mass Balance.
 - b. Energy Balance.
 - c. Simple Composition.
 - d. Custom Calculations.
2. **Abnormal Situation Prevention** – the interface includes:
 - a. Using improved control calculations generate alert messages.
 - b. Integrate specialized calculations from other Emerson Process Management divisions.
 - c. Integrate specialized calculations from 3rd parties or customers.
3. **Reduce project engineering effort** – the reduction in engineering efforts is achieved for several reasons:
 - a. IO and Process checkout.
 - b. Elimination of duplicate display and simulation configuration when engineering systems with simulation requirements.
4. **Provide a framework for Operator Training** – the framework includes:
 - a. Smart process graphics that can be used for process simulation and process calculations.
 - b. Process Graphics include dynamics from both Control Environment as well as Process Modules.
5. **Improve integration with High Fidelity Simulation applications** – the improvements over what is in-place today includes:
 - a. Interface for exchanging runtime configuration. Included in the exchange will be parameter reference information and data type information.
 - b. Interface for exchanging Operator Commands.

2.2 Process Graphic Elements

A set of predefined graphic elements will be provide for the construction of operator displays. These elements may be designed to dynamically show on-line measurements and actuators that interface to the DeltaV control system. In addition, unmeasured parameters that reflect process operation may be calculated using on-line process simulation and may be shown as an integral part of the associated graphic elements.

In an offline environment used for engineering or training environment, the graphic elements will automatically show measurement values base on process simulation. These values calculated by process modules may be based on the actuator position or state as well as manual disturbance values. Process modules will be automatically created from the graphic elements used to construct the operator displays.

The static portion of the graphic elements will in many cases appear similar to the 3D components included in the ifix graphics library. In this section, details will be provided on this library of graphic elements, information displayed with these elements and their link to the control system I/O and process simulation modules.

2.2.1 Process Connections

In many processing plants, the step associated with manufacturing may involve handling solid materials, liquid and vapor, and gases. To clearly illustrate the material flow through the process, three different types of process connections will be supported.

- Piping – for liquid and high pressure vapor or gas flow
- Duct – for low pressure gas flow
- Conveyor – for the movement of solid material between processing units.

The properties of the material that is transferred by a connection are determined by the upstream input. This information plus the *connection status* are available as properties of the connection. The elements that may provide this upstream input are the following:

- Processing element *Output*
- Actuator element *Output*
- Stream element *Output*

The properties of a connection will be automatically displayed when the cursor is placed over the connection. Also, the properties associated with a connection may be exposed for permanent display by placing a measurement or estimated property element on the connection – see section on measurement and estimated property elements.

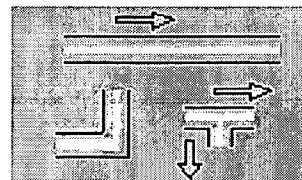
A connection element may terminate at one of the following elements:

- Processing element *Input*
- Actuator element *Input*
- Stream element *Input*

A connection may be created by holding the left mouse button down over a element *Output* and while holding down the mouse position the cursor over an element *Input*. For the connection to be established successfully, the *Input* and *Output* types (pipe, duct, or conveyor) of the upstream and downstream elements may match. The connection will automatically take on the type of the upstream element.

2.2.1.1 Piping

The flow of liquid, high pressure vapor, or high pressure gas between processing elements will be shown as a pipe connection. Piping connections will automatically be routed between processing elements and arrows displayed outside the pipe to show the direction of the flow. If an upstream *Output* is common to two connections, then depending on the placement of the downstream elements, a common line will be shown for part of the connection and branching automatically shown as a “T” in the pipe, as illustrated below.



The pipe connection element will dynamically reflect the following:

- Connection lost – color change.
- Selected property is outside configured limits –color change

Based on the process simulation, the following calculated parameters may be exposed:

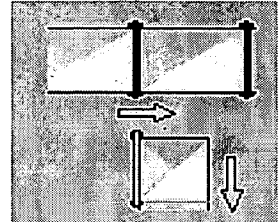
- Properties provided by the upstream connection
- Connection status – good/bad

The following may be configured for this element:

- Limits on selected parameter, color change if outside of limit

2.2.1.2 Air/Gas Duct

The flow of low pressure vapor or gas between processing elements will be shown as a duct connection. Duct connections will automatically be routed between processing elements and arrows displayed outside the duct to show the direction of the flow. If an upstream Output is common to two duct connections, then depending on the placement of the downstream elements, a common duct will be shown for part of the connection and branching automatically shown as a “T” in the duct, as illustrated below.



The duct connection element will dynamically reflect the following:

- Connection lost – color change.
- Selected property is outside configured limits –color change

Based on the process simulation, the following calculated parameters may be exposed:

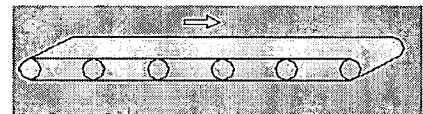
- Properties provided by the upstream connection
- Connection status – good/bad

The following may be configured for this element:

- Limits on selected parameter, color change if outside of limit

2.2.1.3 Conveyor

Solid materials are used or produced by many processes in Industry. Examples are in the pulping process wood chips are transferred from the woodyard to the digester house. In the power house, solid waste material such as bark may be used by the boiler. Plastic pellets are the feedstock to an extruder. The final product of a spray dryer or kiln is a solid material. Ore processing in metals and mining makes extensive use of conveyors.



The material flow along a conveyor is determined by the motor drive connected to the conveyor. Thus, a motor drive actuator may be connected to a conveyor – see motor drive actuator.

Conveyor connections will automatically be routed between processing elements *inputs* and *outputs* defined for conveyor connection. Arrows displayed along the conveyor to show the direction of the flow. The color of the conveyor will change to indicate its status i.e. running or stopped. Also, by connecting a measurement element to the conveyor, it is possible to expose measurements associated with the conveyor e.g. speed or with the material e.g. moisture or weight. Also, the expose property element may be added to the conveyor to display properties that are not measured e.g. composition.

The conveyor connection element will dynamically reflect the following:

- Connection lost – color change.
- Selected property is outside configured limits –color change

Based on the process simulation, the following calculated parameters may be exposed:

- Properties provided by the upstream connection
- Connection status – good/bad

The following may be configured for this element:

- Limits on selected parameter, color change is outside of limit

2.2.2 Process Actuators

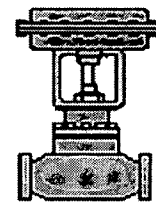
Actuators may be placed between process connects or between processing elements and a connection. In some cases an actuator may be used with a specific connection type i.e. pipe, duct or conveyor, as defined below:

	Pipe	Duct	Conveyor
Regulating valve	X		
On-Off Valve	X		
Pump	X		
Eductor	X		
Force Draft Fan		X	
Induced Draft Fan		X	
Damper Drive		X	
Feeder	X		X
Motor Drive			X

2.2.2.1 Regulating Valve (with actuator)

The regulating valve element will dynamically reflect the following:

- Implied valve position – animation.
- Valve full open/closed – color change
- AO setpoint, PV, OUT mode – numeric, string



Based on the process simulation, the following calculated parameters may be exposed:
Discharge pressure

- Mass flow
- Liquid temperature
- Liquid composition
- Discharge pressure

The following may be configured for this element:

- Reference to AO associated with the valve

- Valve type -linear, quick opening, equal percentage (Sim FB)

2.2.2.2 On-Off Valve (with Actuator)

The on-off valve element will dynamically reflect the following:

- Valve full open/closed/transition – color change
- Valve fail - color
- DO or DC setpoint and mode –string

Based on the process simulation, the following calculated parameters may be exposed:
Discharge pressure

- Mass flow
- Liquid temperature
- Liquid composition
- Inlet pressure

The following may be configured for this element:

- Reference to DO or DC block associated with the valve
- Stroke time from close to open (Sim FB)



2.2.2.3 Manual Valve (Instructor Layer Only)

Manual valves are typically located upstream and/or downstream of regulating valves. Since manual valves directly impact the operation of the process, these elements may be included in the process simulation. To allow local valves included in the simulation to be adjusted by the training instructor during off-line simulation, a graphic element will be provided to represent the manual valve. This graphic element may be included in the instructor layer of the graphic display. The state of the valve (open/closed) will be indicated by the color of the valve.



2.2.2.4 Pump (with Motor)

The pump element will dynamically reflect the following:

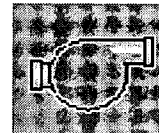
- Motor status – color change
- DC mode and setpoint - strings
- Motor speed (if variable speed drive) – numeric
- AO setpoint, PV, OUT mode (if variable speed drive) – numeric, string

Based on the process simulation, the following calculated parameters may be exposed:

- Discharge pressure
- Liquid composition
- Liquid temperature
- Mass flow

The following may be configured for this element:

- Reference to associated DC block for motor start/stop
- Reference to AO for variable speed drive(if used)

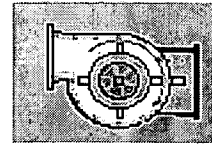


- Pump curve - pressure vs flow (Sim FB)

2.2.2.5 Force Draft Fan(with Motor)

The force draft fan element will dynamically reflect the

- Motor status – color change
- DC mode and setpoint - strings
- Motor speed (if variable speed drive) – numeric
- AO setpoint, PV, OUT mode (if variable speed drive) – numeric, string



following:

Based on the process simulation, the following calculated parameters may be exposed:

- Discharge pressure
- Gas composition
- Gas temperature
- Gas mass flow

The following may be configured for this element:

- Reference to associated DC block for motor start/stop
- Reference to AO for variable speed drive(if used)
- Fan curve - pressure vs flow (Sim FB)

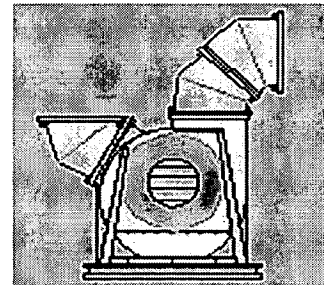
2.2.2.6 Induced Draft Fan(with Motor)

The induced draft fan element will dynamically reflect the following:

- Motor status – color change
- DC mode and setpoint - strings
- Motor speed (if variable speed drive) – numeric
- AO setpoint, PV, OUT mode (if variable speed drive) – numeric, string

Based on the process simulation, the following calculated parameters may be exposed:

- Gas composition
- Gas temperature
- Gas mass flow

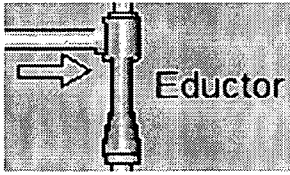


The following may be configured for this element:

- Reference to associated DC block for motor start/stop
- Reference to AO for variable speed drive(if used)
- Fan curve - pressure vs flow (Sim FB)
-

2.2.2.7 Eductor (with On-off Valve)

1. xyz
 - a. abc



2.2.2.8 Damper (with Drive)

2. xyz

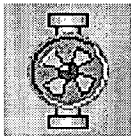
b. abc



2.2.2.9 Feeder (with variable speed Motor)

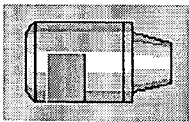
3. xyz

c. abc



2.2.2.10 Conveyor Motor Drive

The motor drive may be attached to a conveyor element



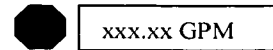
2.2.3 Measurements and Property Elements

The transmitter element is used in the display to access the measurement value associated with a physical transmitter or switch. This element can be added to a connection element or to a processing element. When a transmitter element is added to the display, the user may identify the associated AI, PCI or DI block in DeltaV. In the on-line mode, the value of the measurement will be shown next to this measurement element. In the off-line mode the simulated value of the measurement will be automatically displayed based on the value calculated by the associated process simulation module.

An estimated property element may be added to a connection or processing element to display any property of that element. When this element is placed on a connection or on a piece of equipment then the user can browse and select the properties that will be displayed. Thus, properties that are not available through a physical measurement may be exposed through the use of the estimated properties element.

In this section, we discuss the elements provided for process measurement and for accessing estimated properties calculated by the process simulation modules.

2.2.3.1 Transmitter Element



The transmitter element will dynamically reflect the following:

- Bad or uncertain status – color change
- Mode of associated AI (Manual or OS) – color change
- Measurement value and units – numeric and string

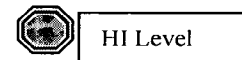
When in off-line simulation mode, the following parameters may be access and changed:

- Select simulation value or manual value and status for display - boolean
- Manually entered value and status of measurement

The following may be configured for this element:

- Reference to associated AI or PCI block for measurement
- Reference to the simulated property that this measurement reflects (Sim FB)

2.2.3.2 Physical Switch



The physical switch element will dynamically reflect the following:

- Bad or uncertain status – color change
- Mode of associated DI in Manual or OS – color change
- Switch discrete value–string

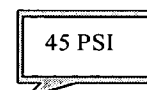
When in off-line simulation mode, the following parameters may be access and changed:

- Select simulation value or manual value and status - boolean
- Manually entered value and status of switch

The following may be configured for this element:

- Reference to associated DI block
- Reference to the element property that triggers the switch (Sim FB)
- The limit and deadband associated with change of state (Sim FB).

2.2.3.3 Estimated Property



The estimated property element will dynamically reflect the following:

- Good/Bad connection – color change
- Estimated property value(s) – numeric
- Property outside limit – color change

The following may be configured for this element:

- Reference to property(s) to be displayed (Sim FB)
- Limits and color change if outside of limits (Sim FB)

2.2.4 Processing Elements

Plant equipment other than measurements and actuator and connection will be represented in the operator display as processing elements. All inputs and output to a processing elements will be made through connection elements. Standard processing elements will be provided for the following plant equipment:

- Tank – vertical
- Tanks – horizontal
- Heater
- Static Mixer
- Reactor
- Mixer
- Air Heater

For these standard processing elements, the user may specify the number of inputs and outputs to and physical equipment properties e.g. size. The simulation algorithm and static representation of these standard processing elements may not be modified by the user.

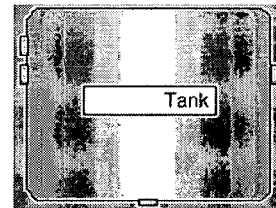
All other plant equipment will be implemented as custom processing elements. The static representation, number of inputs and outputs and the simulation algorithm may be modified to meet the user interface requirements. Once a custom processing element has been defined, it may be saved as a template that may be reused or used as a starting point in the creation of other processing elements.

2.2.4.1 Standard Processing Elements

2.2.4.1.1 Tank – Vertical

Based on the pipe connections to the vertical tank, the tank element will dynamically reflect the following:

- Level in the tank – dynamic animation
- Level at 100% or empty – color change



Based on the process simulation, the following calculated parameters may be exposed:

- Outlet temperature
- Outlet composition
- Liquid temperature
- Simulated level

The following may be configured for this element:

- Number of input and output connections
- Complete connections to the tank
- Tanks Diameter and height (Sim FB)

2.2.4.1.2 Tank - Horizontal

Based on the pipe connections to the vertical tank, the tank element will dynamically reflect the following:

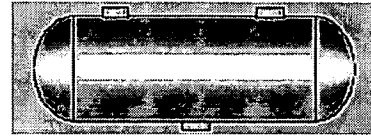
- Level in the tank – dynamic animation
- Level at 100% or empty – color change

Based on the process simulation, the following calculated parameters may be exposed:

- Outlet temperature
- Outlet composition
- Liquid temperature
- Simulated level

The following may be configured for this element:

- Number of input and output connections
- Complete connections to the tank
- Tanks Diameter and length (Sim FB)



2.2.4.1.3 Heater

Based on the pipe connections to the heater, the tank element will dynamically reflect the following:

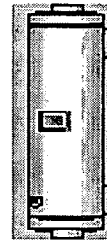
- Low heat transfer coefficient – color change

Based on the process simulation, the following calculated parameters may be exposed:

- Heat transfer coefficient
- Outlet product temperature
- Inlet product temperature
- Outlet pressure (assuming fixed drop)

The following may be configured for this element:

- Complete connections to the heater
- Heater surface area (Sim FB)
- Heat transfer coefficient when clean (Sim FB)



2.2.4.1.4 Static Mixer

1. xyz

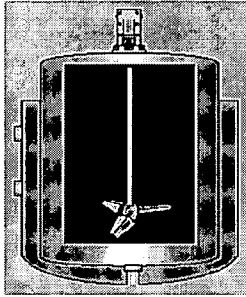
a. abc



2.2.4.1.5 Reactor

2. xyz

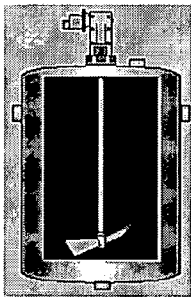
b. abc



2.2.4.1.6 Mixer

3. xyz

c. abc



2.2.4.1.7 Air Heater

4. xyz

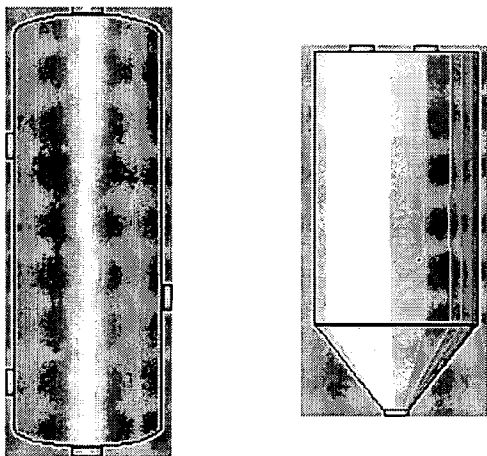
d. abc



2.2.4.2 Custom Processing Element

Many processing units such as distillation columns, evaporators, separators, boiler etc many be represented graphically as a processing element. The simulation associated with the vessel may be user defined if not included in our standard selection. Normally the processing will be described as the step response model relating each input to each output of the vessel. Inputs may be gas and/or liquid streams. Optionally, the user may define the equations that describe the relationships between the inputs and outputs of the processing element.

To help users quickly create the static graphics associated with a custom processing element, some simple static graph representation will be provided that the user may select. If these simple graphics are used, then the user may specify the desired number of input and output connects and the type of connection supported i.e. pipie, duct, or conveyor. In response, the graphic item will be displayed and can be immediately used in the creation of the operator graphic



The gains any dynamics associated with each input and output of the process element may be specified as show below if the user elects to specify the simulation algorithm as step responses:

<div><input type="radio"/> Input 1</div> <div><input checked="" type="radio"/> Input 2</div> <div><input type="radio"/> Input 3</div>	<table border="1"><thead><tr><th></th><th>Gain</th><th>Deadtime</th><th>Time Constant</th><th>Bias</th></tr></thead><tbody><tr><td>Output 1</td><td>1.5</td><td>5</td><td>30</td><td>16</td></tr><tr><td>Output 2</td><td>0</td><td>0</td><td>0</td><td>55</td></tr></tbody></table>		Gain	Deadtime	Time Constant	Bias	Output 1	1.5	5	30	16	Output 2	0	0	0	55
	Gain	Deadtime	Time Constant	Bias												
Output 1	1.5	5	30	16												
Output 2	0	0	0	55												

If the user selects custom algorithm, then an expression editor similar to that used with the Calc/Logic block will be provided for the user to define the simulation algorithm. Based on the method selected, the properties of the custom processing element outputs may be calculated.

By attaching transmitter elements to a custom processing element, the properties associated with the process element inputs or outputs may be referenced for off-line simulation. These properties may also be made visible in the display using an estimated property element.

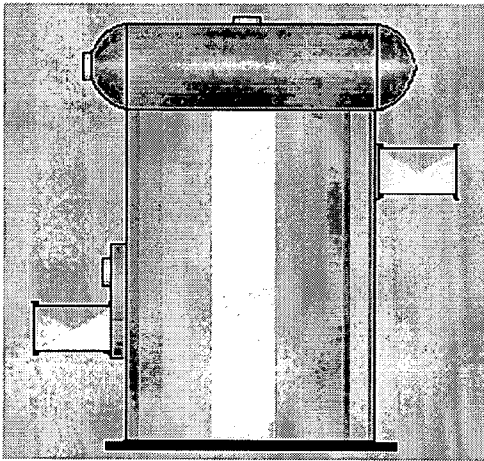
Several pre-defined templates will be provided for custom processing elements. These templates will be described in this section.

2.2.4.2.1 Boiler Template

The following static graphic will be provided for the creation of a custom processing element that represents a boiler. As part of the template, a custom algorithm will be provide that will calculate the following process outputs:

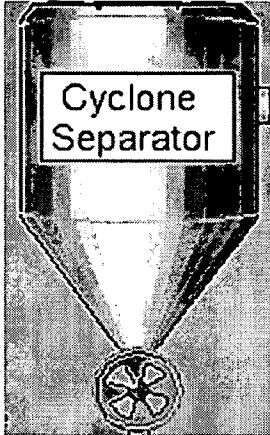
- Exit gas O₂
- Exit gas CO
- Steam generated
- Boiler drum level
- Boiler draft

The template will be based on a single fuel input. By modifying this template, it will possible to simulate boilers with multiple fuels.



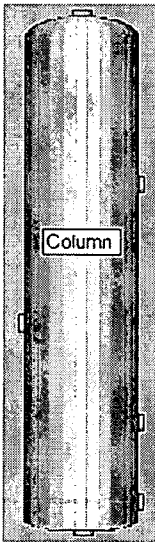
2.2.4.2.2 Specialize Vessel –Cyclone Separator

The cyclone separator may be used with in conjunction with the spay dryer custom processing element. A step response model may be used to define the expected response.



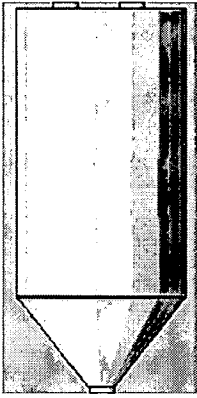
2.2.4.2.3 Column

The column custom processing element will utilize a step response model to define the expected process response.



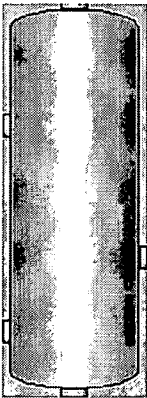
2.2.4.2.4 Specialize Vessel –Spray Dryer

The spray dryer custom processing element will utilize a step response model to define the expected process response.



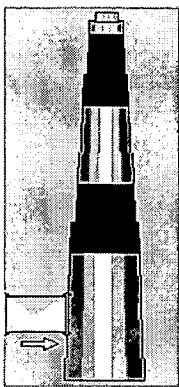
2.2.4.2.5 Specialize Vessel –Evaporator Body

The evaporator custom processing element will use a custom model to define the expected process response. Based on the energy input and the concentration of the input flow, the concentration of the outlet flow and vapor release will be calculated. Multiple evaporator elements may be connected together along with heat exchanger and eductor elements to create a multiple-effect evaporator.



2.2.4.2.6 Specialize Vessel -Stack

The stack custom processing element may be used with the boiler processing element. The properties of inlet are carried through the stack with no modifications.



2.3 Process Graphic Streams

Stream elements may be included in a display to define the starting properties associated with a connection element. Also, stream elements may be used as connection points between displays. For such off-sheet connections between displays, the user may click on the stream to immediately call up the associated display that contains the referenced connection.

2.3.1 Mass / Composition

The mass/composition stream element will normally be used for define the starting properties of a process input i.e. the starting feedstock composition, etc. or to define a link to a stream connection on another display. Connections may be made on the input or output of the mass/composition steam element (not both)



The following may be configured for this element:

- Name of the stream (unique within the DeltaV System)
- Properties of the steam (if no reference input of input connection)
 - Mass fraction of component (if more than one)
 - Pressure or mass flow
 - Temperature
 - Specific heat
 - Density
 - Connection type (pipe, duct, conveyor)
- Referenced Input stream(if used of accessing a stream on another display)

2.3.2 Energy

The energy stream element will normally be used for define the starting energy associated with a process input i.e. the BTU/HR transfer, etc. or to define a link to the energy properties of stream connection on another display. Connections may be made on the input or output of the energy stream element (not both)



The following may be configured for this element:

- Name of the stream (unique within the DeltaV System)
- Properties of the steam (if no reference input of input connection)

- Energy flow
- Connection type (pipe, duct, conveyor)
- Referenced Input stream(if used of accessing a stream on another display)

2.4 Local Panels and Subsystems (Instructor Layer Only)

In some process areas of a plant, a local panel will be installed to allow local start and stop of motors, to provide local indications of process conditions e.g. local temperature/level indicator, etc. To allow this interface and any associated indications to be included in the training system, a Local Panel graphic element may be associated with an actuator element. This graphic element for a local panel may be included in the Instructor lay of the operator display. It is envisioned that a set of local panel graphic elements will be created to address the different types of standard actuator elements.

Some pieces of equipment such as compressors, generators, etc. may be purchased and supplied with instrumentation, specialized control, and in some cases local indications for temperature, level, etc. Such subsystems may in some cases be interfaced to the DeltaV system through the serial interface card, ASI bus interface, etc. The associated process equipment and local controls and indications may be simulated using a custom process element. A custom process element may be created to represent the local panel interface to the control and indicators. This subsystem interface may be added to the Instructor layer of the operator display.

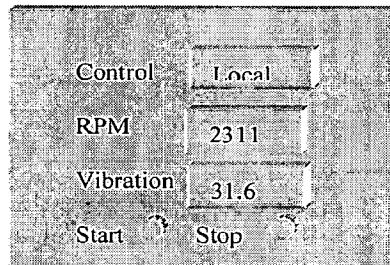
2.4.1 Standard Actuators

The standard actuator elements that include a motor or on/off actuator will be designed include the capability to simulate the HOA function. A output indicating local operation will supported so this may be referenced by a DI block that provides the Track input to the DC block. When the training instructor clicks on the graphic element for a standard actuator, then a standard faceplate will be presented that allows the instructor to select hand, off or auto as illustrated below. As an option, this faceplate may be docked and included in the display at an instructor station.



2.4.2 Subsystems

The local control logic associated with a subsystem may be included in the simulation function block algorithm associated with the process equipment. Since the inputs and outputs included on a local panel are specific to the subsystem and to the process equipment, a custom graphic element for the local panel may be constructed and included in the Instructor layer of the operator display. An example of a custom graphic element for a local subsystem panel is shown below:



2.5 Process Modules / Simulations

Process simulation function blocks will be provided for each element (see section 2.3) provided for the creation of a graphic display. Based on the elements contained in a display and their associated connection, these simulation function blocks will automatically be placed in a process module that will be used to simulate the process. The basis of the process simulation is:

simulation of the process will automatically be created using these simulation function block

2.5.1 Actuator Elements

The actuator element simulation will be based on the behavior of the mechanical device used for actuation. In the case of motors, the DC definition will be automatically utilized in the simulation. Dynamics associated with the actuator element will be limited to the stroking time associated with valve.

2.5.2 Processing Elements

The processing element simulation will be based on the defined inputs and outputs and 1) step response or 2) custom calculations based on simple energy and material balances. Thus, the properties that will be available for these blocks in most cases will be limited to:

- Outlet temperature (based on inlet temperature, flow and heat capacity)
- Outlet flow (based on inlet mass flow and accumulation within the element)
- Outlet pressure (based on assumed drop across the unit)
- Outlet composition (based on perfect mixing and inlet composition)

When custom calculations are implemented, then the built-in dynamics associated with the outlet properties will added be based on 1st order plus deadtime response to changes in the process inputs. The user may specify the deadtime and lag associated with each calculated property.

2.5.3 Process Measurement Elements

It will be assumed that the transmitter and switch introduce no dynamics in the referenced property.

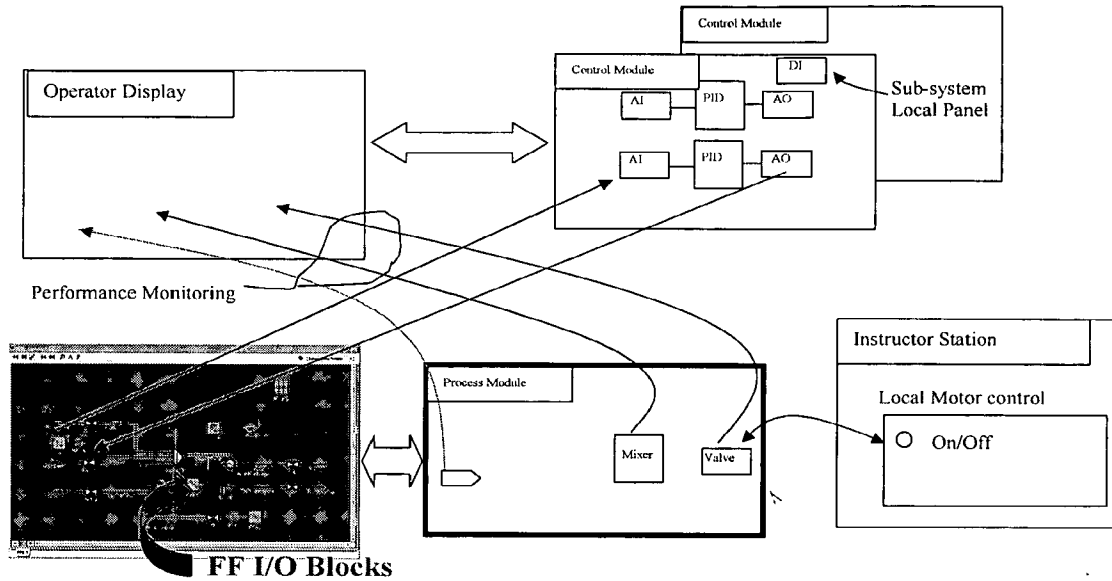
2.5.4 Connections

It will be assumed that no dynamics are introduced by a connection i.e. the properties from the upstream connection are immediately reflected in the downstream connection.

2.6 High Fidelity Simulations

2.6.1 General

The I/O references that are defined for measurement and actuator elements will be used to automatically create the is used by HYSYS. Standard processing element templates will be defined for each HYSYS component that may be used to construct a high fidelity process simulation. When these templates are used with HYSYS, then the associated simulation function block in the process module act as shadow blocks i.e. their process simulation algorithm is not executed and the block parameters are read and written by HYSYS.



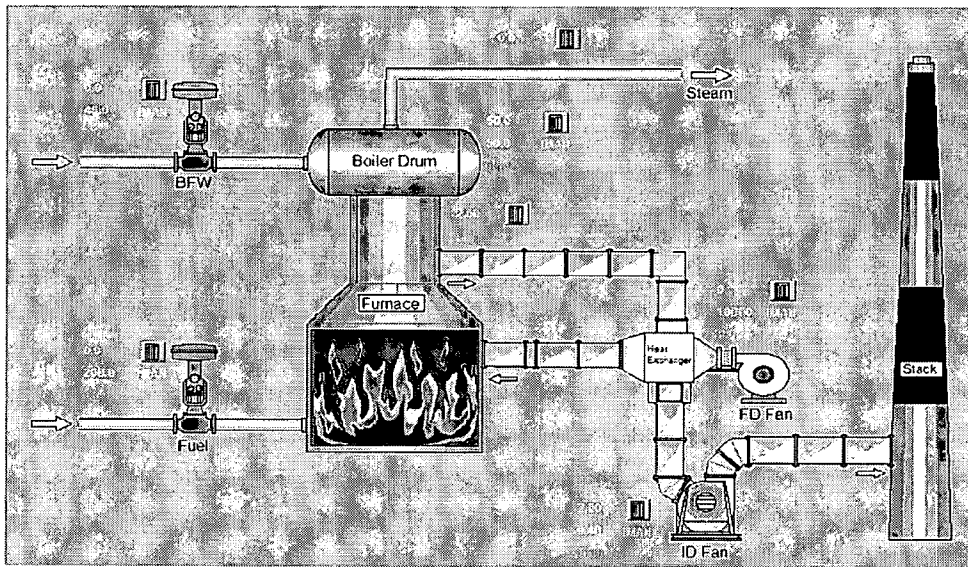
2.6.2 Operator Training

The Instructor layer added to the operator display will provide instructor access to the manual actuators, and interfaces provided at local panels for actuators and subsystems. This capability will eliminate the requirement to implement separate instructor displays in training systems. Also, the dynamic simulation provided for the DC and DO blocks will eliminate the need to add low level logic to simulate the physical contacts provided by actuators and valves. Thus, the high fidelity simulation used for operator training could focus on the critical unit operations within the plant.

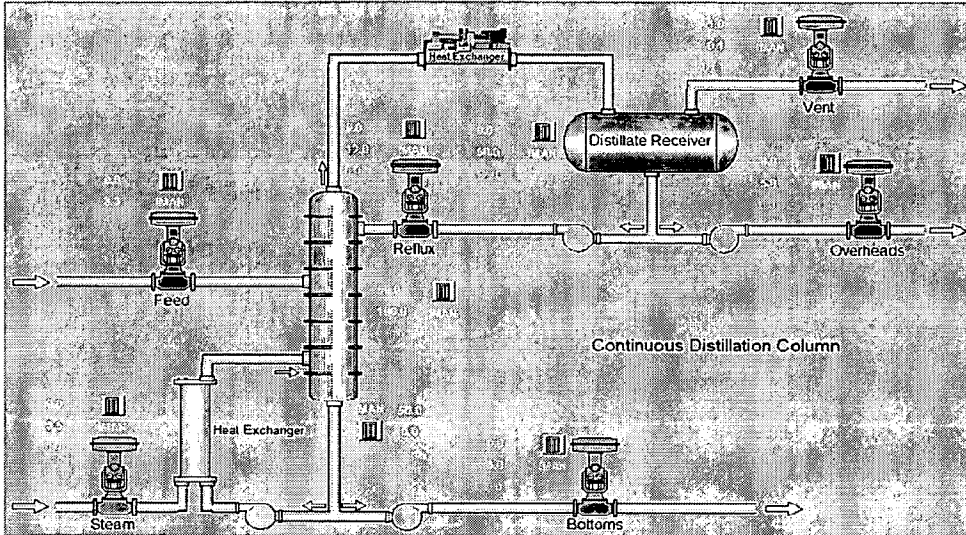
2.7 Example Displays

The following examples illustrate how the graphic elements defined in this document may be used to create graphic displays.

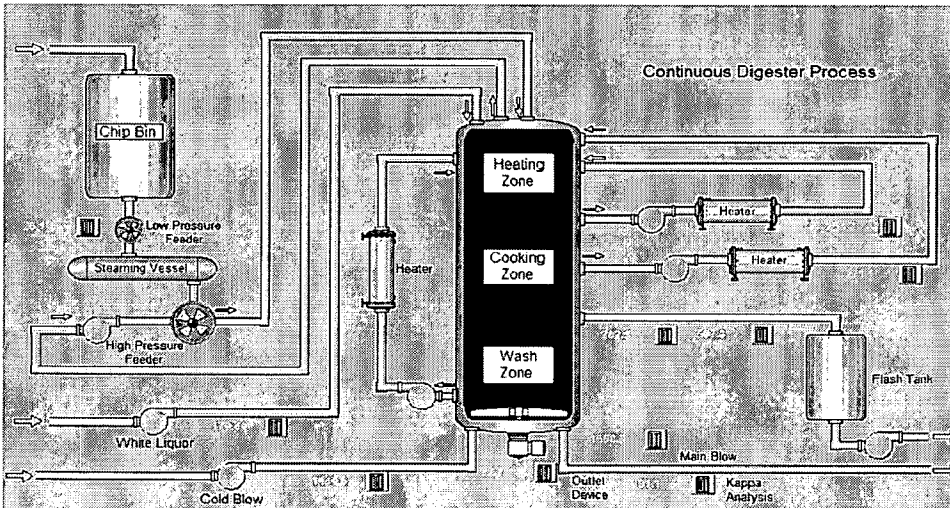
2.7.1 Boiler Example



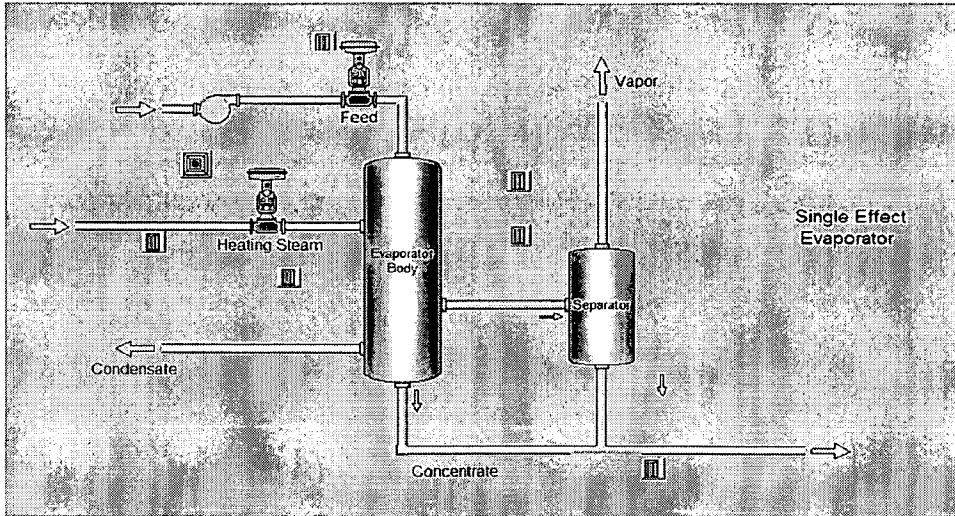
2.7.2 Column Example



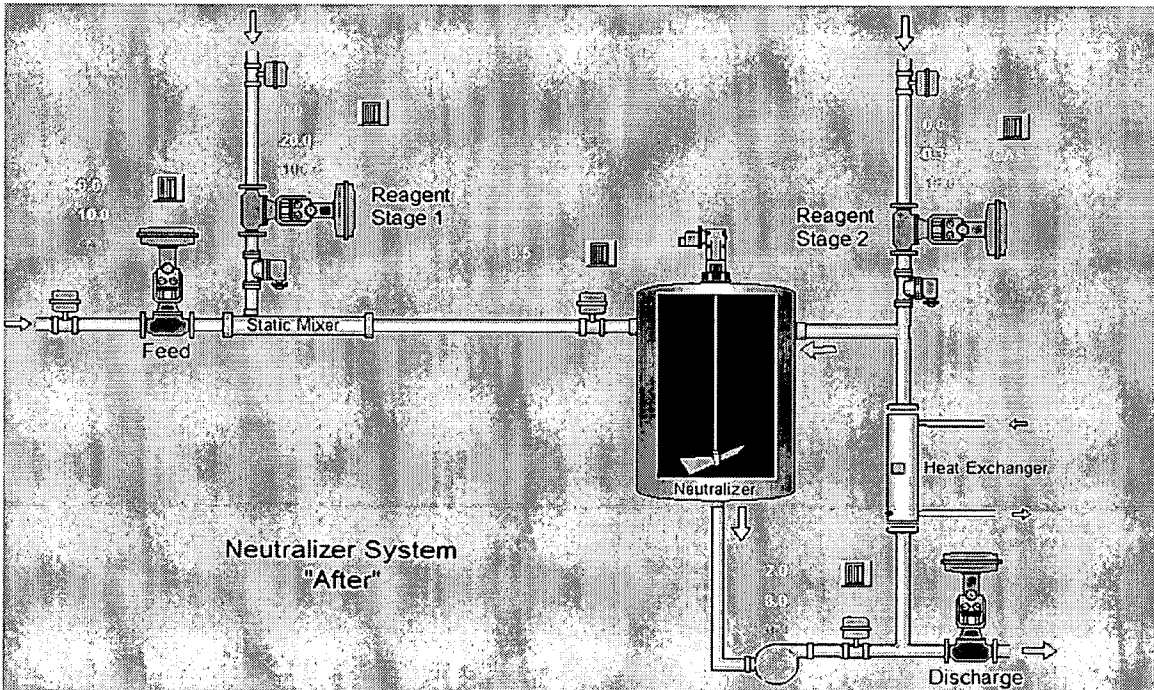
2.7.3 Custom Processing Example – Kamyir Digester



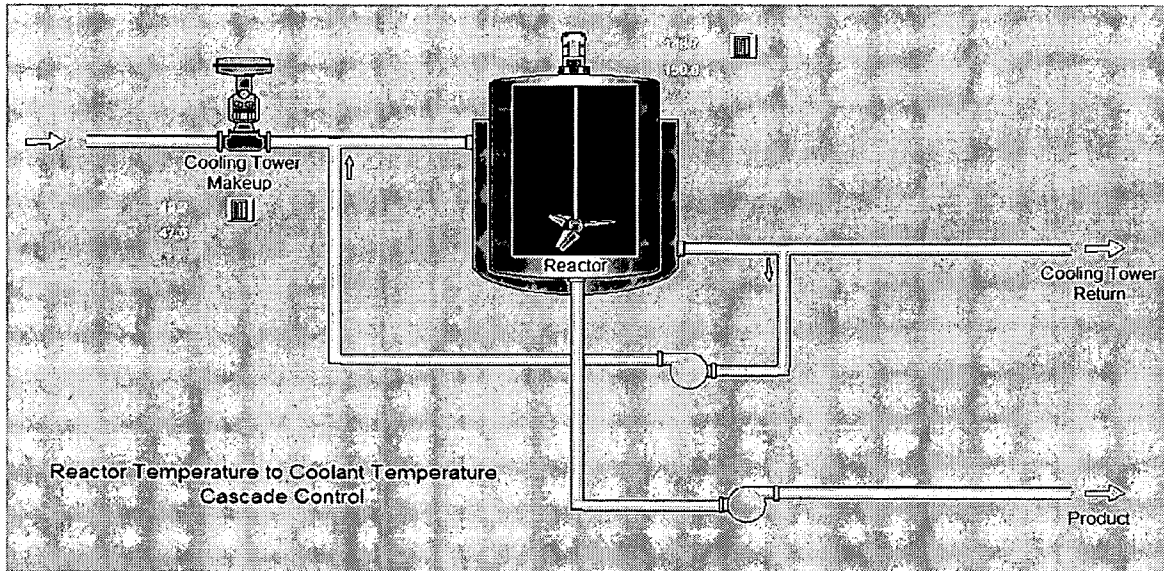
2.7.4 Evaporator Separator Example



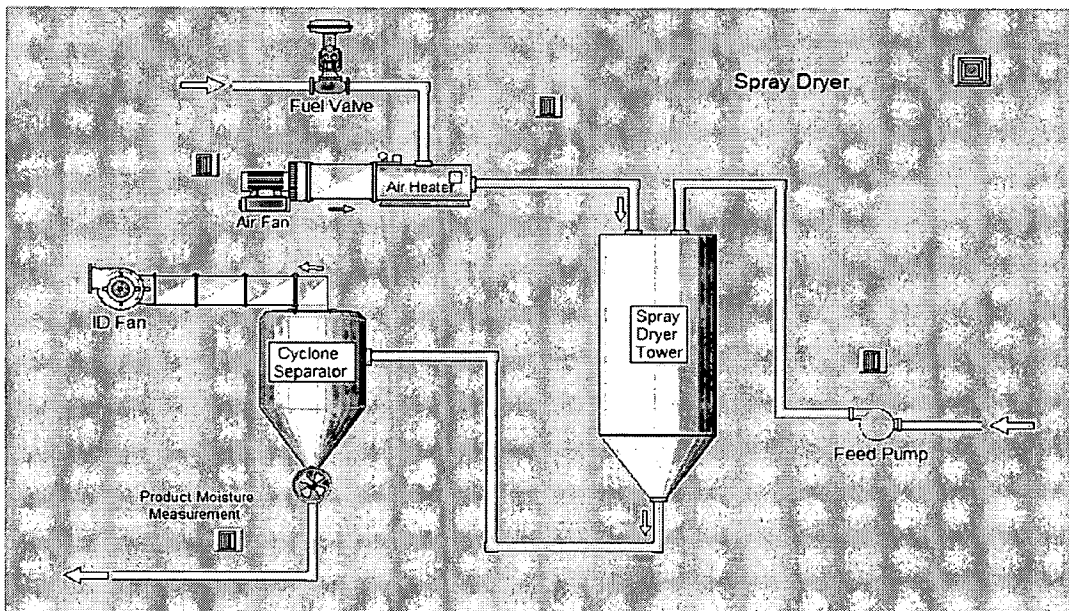
2.7.5 Mixer Heater Example



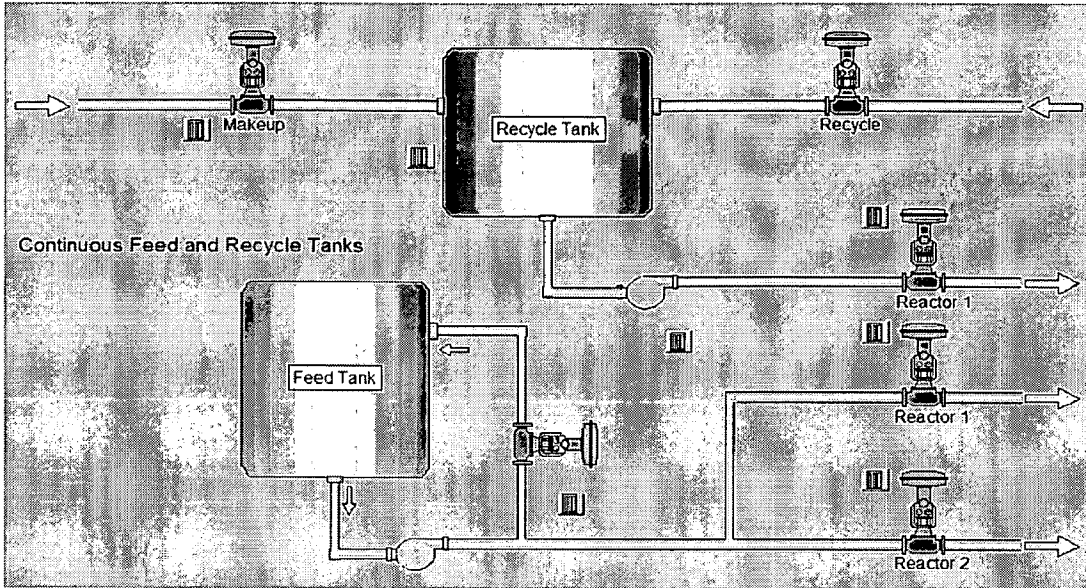
2.7.6 Reactor Example



2.7.7 Spray Dryer Cyclone Separator Example



2.7.8 Tank Example



3 Concept

This section discusses the design of Process Graphic Displays and process modules. Also, information will be provided on how a process module may be automatically created from a graphic display or manually constructed in control studio. Details are provided graph elements and simulation function blocks used to create displays and simulation modules. Also, information is provided on how high fidelity process simulation packages, such as HYSYS, may be linked into DeltaV displays and control using the process modules.

3.1 Overview

At the Process Graphic level we have a view that is very similar to what is available with iFIX today. At this level the user selects Process Graphic elements such as Valve, Pump, Tank, Heater, Mixer, etc drops them on a Process Graphic and then connects them together using piping. The user adds animations, dynamic values, etc to control, measurements or actuators as they would with iFIX.

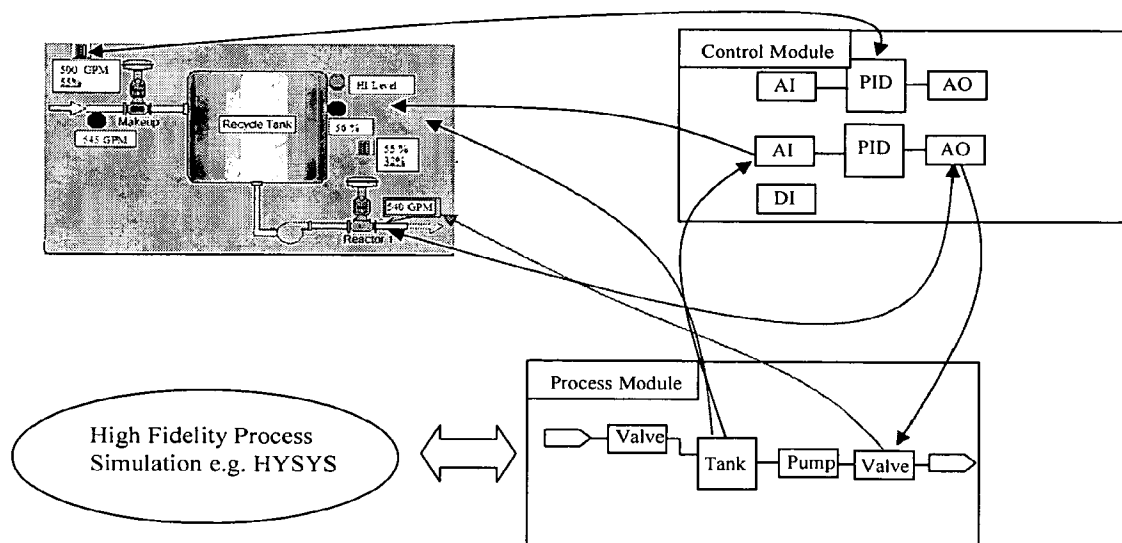
The user can also select the Process Graphic Display and enable process simulation. The dynamic simulation will be constructed from simulation function blocks that are based on simple composition calculations, mass balance, energy balance and custom calculations.

If high fidelity process simulation is required, then a simulation package such as HYSYS may be linked into DeltaV through process modules. In this manner, information from the process simulation may be made available on the Process Graphic Display.

A Process Module may be automatically generated from a process graphics. The functionality available to the Process Module is determined by the Process Graphic elements. What should be clear is that the Process Module will be constructed to shadow the Process Graphic Display.

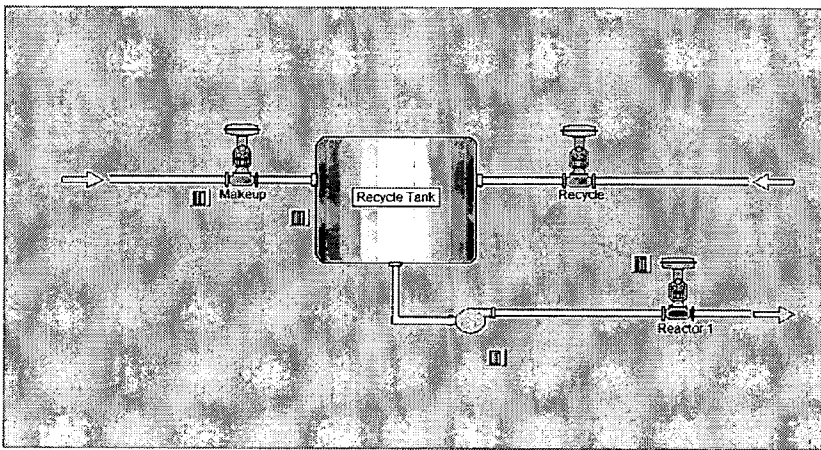
When the user configures their Process Graphic Display they have the ability to include additional information such as Mass or Energy Streams. These streams are used in the process module to establish starting conditions needed by the simulation function blocks.

Since Process Modules are real DeltaV Modules, it is also possible for them to reference, and be referenced by, DeltaV parameters, control strategies, displays, etc as illustrated below. Also, using this capability, it is possible for a process module to be created in DeltaV Control Studio independent of the Process Graphic Display.

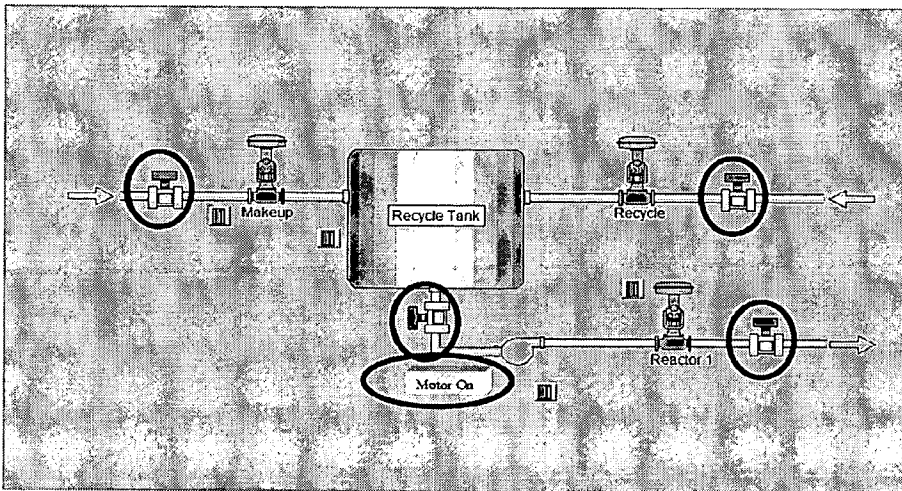


3.1.1 Display Layers – Off-line Simulation

The process simulation may be designed to include hand valves and local panel inputs that will exist in the plant but are not wired into the DeltaV control. Traditional operator training systems(OTS) have traditionally included custom built instructor graphic displays that included graphic elements to represent and provide access to these local component. To avoid the need for separate instructor displays, the operator graphic will be designed to support layers of information that may be visible or not visible depending on the user and whether the display is being used for off-line simulation or on-line operation. In particular, it will be possible for the user to add an instructor layer to any operator graphic. Through this instructor layer, it will be possible to add manual valves and other graphic elements that represent and provide access to the simulation of local valves, local panel input, and disturbance inputs to the process simulation. For example, the operator graphic for on-line operations might appear as shown below.



In the off-line environment, the display would automatically display manual valves and local panel inputs if the a user at the associated station has "Instructor" privilege. This added layer of information would be show with the other information normally presented to the operator as shown below.



If the instructor clicks on the manual valve or local panel graphic element, then a faceplate will be presented from which the instructor may make changes e.g. open close manual valve, etc.

Also, in the off-line simulation, the instructor layer will also add behavior to the I/O elements i.e. transmitter, motors, damper or valve element included in the display. In response to the instructor clicking on one of these graphic elements, a faceplate will be presented to the instructor that allows him to enter disturbances into the process simulation. For example, the faceplate provided for a transmitter would allow the instructor to do one of the following:

- Bias the measurement value
- Add noise to the measurement
- Enter a fixed value for the measurement value
- Change the measurement status from Good, Bad or Uncertain.

The faceplate selections associated with a motorized valve or motor would allow the instructor to fail the motor run feedback, torque limit switch, etc i.e. force not running when the setpoint and associated discrete outputs of the DC block are set to run state.

3.2 Process Graphic Display

In the graphic display editor, a number of display palettes to support the creation of graphic displays. The following standard palettes provided in DeltaV:

- Calculation and Control – elements to access information from function blocks used in control and calculation e.g. PID function block SP, PV, OUT
- Properties and Measurement– elements used to access or specify field measurements and simulated properties associated with connections or process equipment i.e. field measurement, simulated values.
- Actuators – elements representing field devices used to set or regulate process streams
- Processing – elements representing common process equipment.
- Custom – elements that allow equipment that is specific to a manufacturing process to be represented in the graphic display.

As in Control Studio, the user may create his own palette from items included in these standard palettes. All the elements included in the display except for Steam and Connection elements may be modified by the end user to satisfy his display requirement.

3.2.1 Calculation and Control

The elements provided to access function block parameters in control modules will be similar to the module and faceplate dynamo's currently provided in ifix. However, an interactive editor will be provided to make it easier to create or modify the values displayed, associated faceplates and details displays. These elements will reference function block parameters in control modules i.e. no reference is made to the process control module.

3.2.2 Properties and Measurement

The measurement and switch elements may be attached to a connection or processing element to show its associated properties. In the graphic, these will appear similar to the dynamo's, faceplate, and detail display provided in ifix to access measurement values. These elements will reference the associated AI or DI function block. Also, as part of the element configuration, a reference to a property of a processing element or connection can be define. This reference will be used by the associated process module to write this property value to the SIMULATE_IN or SIMULATE_IN_D parameter of the associated AI or DI function blocks. Thus, when simulation is enabled in the AI or DI function block, the simulated value will be used by the function block in place of the measurement. Also, if this reference is configured, then other simulated properties of the valve may be displayed with the valve or used to change color, etc associated with the valve.

The estimated property element will appear in a display similar to measurement elements. However, these elements will reference the simulation function block parameters contained in a Process Module.

3.2.3 Actuator Elements

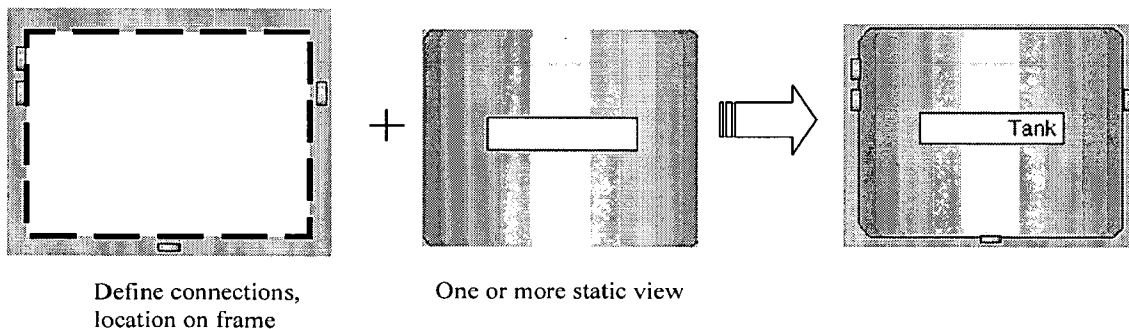
A wide variety of standard actuators will be provided for creation of a display. The graphic representation will reflect the type of actuator. When an actuator element is selected in the display, then a faceplate and detail display provided in ifix to access output blocks. These actuator elements will reference an AO, DO and DC function blocks. Also a reference to a simulation function block may be configured for this element. This reference will be used by the actuator element to display simulated properties of the actuator or used to change actuator color, etc. Also, based on this reference, parameter of the function block will be provided to the simulation function block.

The customer may modify the static view associate with actuator element to give these a custom appearance. Also, this element will have configurable options to change colors based on the actuator properties

3.2.4 Processing Elements

A standard set of processing elements will be provided for common application in the process industry e.g. storage tank. Also, custom process elements will be provide to address application that are unique to a manufacturing process. All process elements and their associated simulation function block will share a common structure.

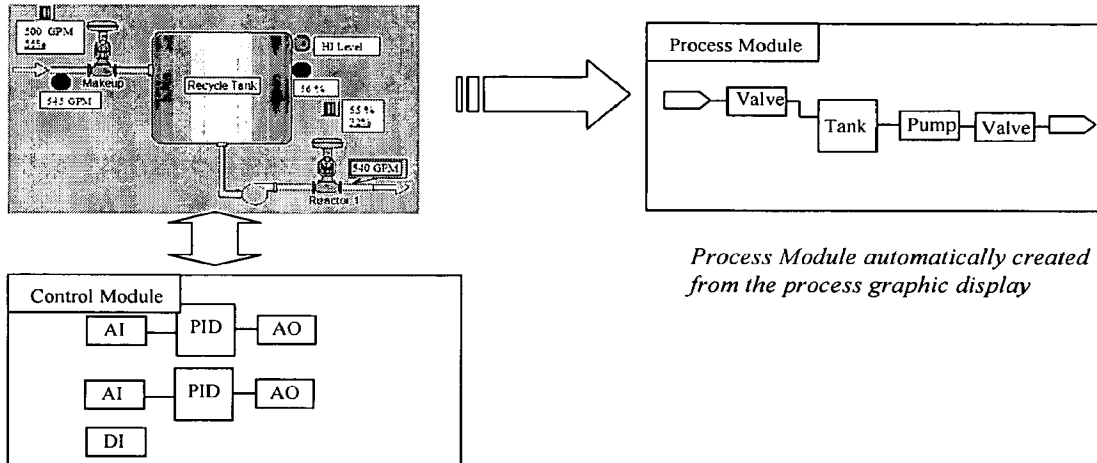
The visual appearance process element in a graphic display is defined by its connections and static view. To define a new processing element, the user will need to specify the number of input and output connections, their relative location and type (pipe, duct, or conveyor) assigned to each connection. This could be done by the user assigning connection items to a basic frame and then specifies one or more static view that will be associated with the graphic representation, as illustrated below:



The static view that is show at any given time may be specified by the user based on a referenced parameter of a function block or a simulation function block. In addition, the capability will be provide to include dymanic elements (values, strings, bars, etc) within the processing element.

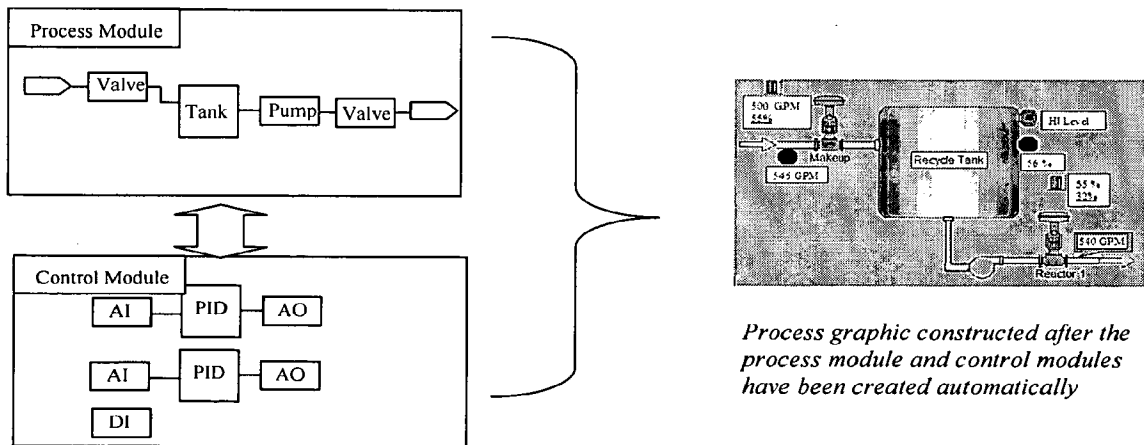
3.3 Process Module

The process modules will consist of simulation function blocks, streams, and their associated connections. Since there is a one to one correspondence between the process graphics elements and simulation function blocks, it will be possible for a customer to construct a graphic display and to automatically generate the corresponding process module, as illustrated below.

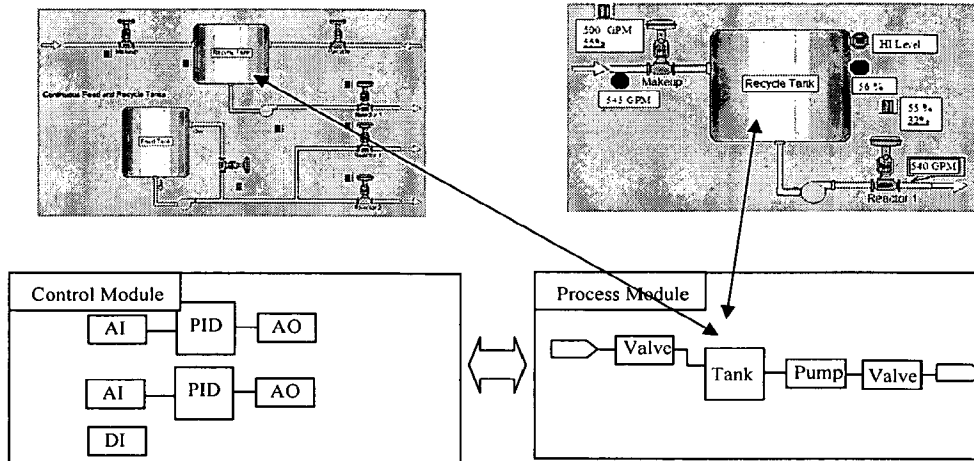


To allow this automatic generation of the process module, it will be necessary for the customer to identify the actuator, connection or processing element property associated with measurement element and estimated property elements.

A customer may need to create a process simulation before process graphics or, in some cases, before the control module are constructed. Thus, the process module type will be supported in control studio. When this type is selected, then the simulation function block palette will be provided to construct a process simulation. After the simulation is constructed, it will be possible to fill in the references to the I/O blocks in the control module. Also, when the associated graphic display is created, it will be possible to browse to the existing process module to set the property references.



In some cases the process graphic may not contain all the detail needed to construct the process simulation. Thus, it is important that the capability be provided to edit and simulation module that have been automatically created from a process graphic. Also, since multiple process graphics may need to display the same piece of equipment, it is necessary in the construction of a process graphic for an element to reference an existing process module.



Properties calculated by a process modules may be included in multiple process graphics

3.3.1 Simulation Blocks

The simulation function blocks that correspond to the processing elements will have a common structure. These block are characterize by the following features:

- The block input connections and the parameters of the block are used in the simulation i.e. no reference to control module is supported.
- The number of inputs and outputs connections supported by the block may be defined as extensible
- Results from the block executing may be reflected in the block output connections or as parameters of the block.
- The block algorithm may be defined as a step response or may be entered by the user similar to the Calc/Logic block. When the algorithm is entered then the user may independently specify the dynamic for each output.
- Local panel or subsystem inputs or logic that is directly associated with a processing element may be included in the block algorithm.
- A common set of parameters will be supported for input and output connection.

3.3.1.1 Input and Output Connections

The parameters associated with input and output connections will be communicated between blocks as an array parameter or structure consisting of the following parameters.

1. Connection (Good/Bad)
2. Mass Flow
3. Pressure
4. Temperature

5. Specific Heat
6. Density

In some cases, other parameters such as composition of a steam is required in the function block algorithm. To support this requirement, a standard and extended Stream block will be provided. As part of the extended stream block configuration, the user may select a set of pre-defined groups of data. These extended connections will be allowed to connect to block that utilize this information. In general, the extended parameters set will consist of the following:

7. Group Name
8. Element 1
9. Element 2
10. Element 3
11. Element 4
12. Element 5.....

One of the pre-defined group sets may be the fuel input to the function block associated with the boiler processing element. For this example, the group would contain the component of the fuel.

7. Fuel Set
8. Carbon, wt %
9. Hydrogen in fuel, wt%
10. Sulfur in fuel, wt %
11. Oxygen in fuel, wt %
12. Moisture in fuel, wt%
13. Nitrogen in fuel, wt %

For a turbogenerator processing element, the connections to the associated simulation block may use an extended parameter set that includes:

7. Steam Set
8. Steam Enthalpy (Actual entering the stage
9. Steam Enthalpy (Actual) exiting the stage
10. Steam Enthalpy (If isentropic expansion)

The expanded group set will also be used when simulation blocks are used as an interface to high fidelity simulation packages where on some streams the composition is to be visible in the process graphic.

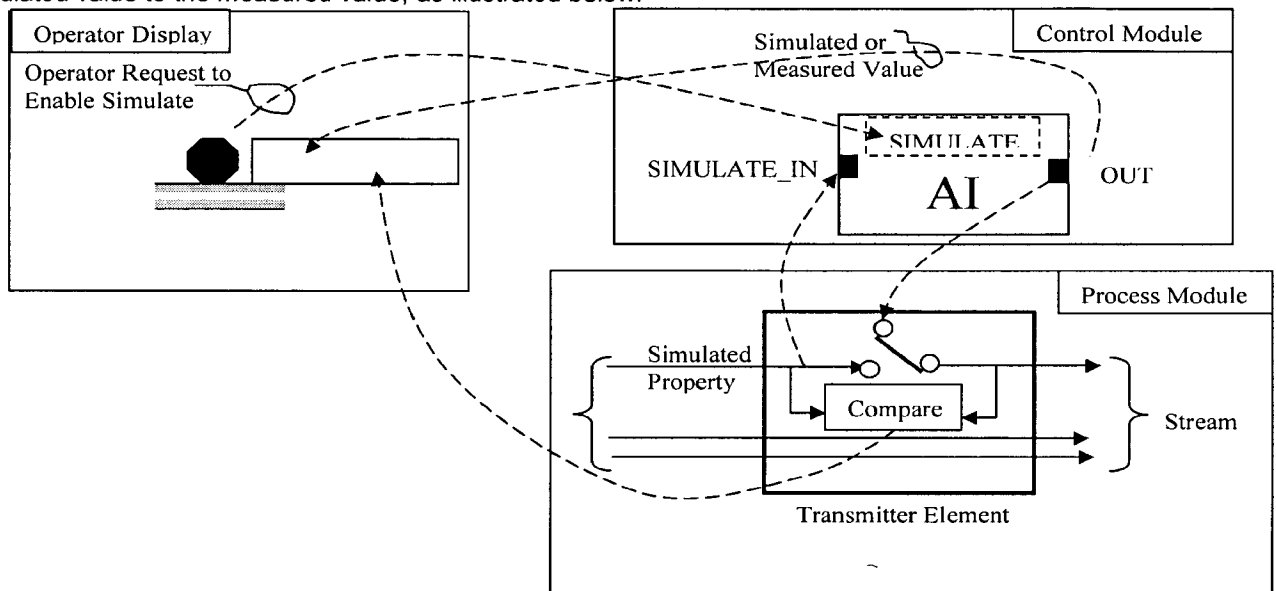
3.3.1.2 DeltaV References

When a measurement or actuator element is added to a process graphic, the user may browse to select the associated I/O block in a DeltaV Control module. During off-line operation, the calculated measurement or measured position from the simulation will be automatically written to the Simulate_IN parameter in the associated input and output function block.

During on-line operation, the measurement value may be utilized in the process simulation. The associated parameter may be compared to the simulated value and used to alert the operator to a difference in these values. When a transmitter measurement or actuator feedback element is detected as failed, then the operator may elect to use a simulated value. This simulated value will be through the Simulate_IN parameter of the associated block(s). In this section, we address the inputs that may be provided to function blocks in a Control module.

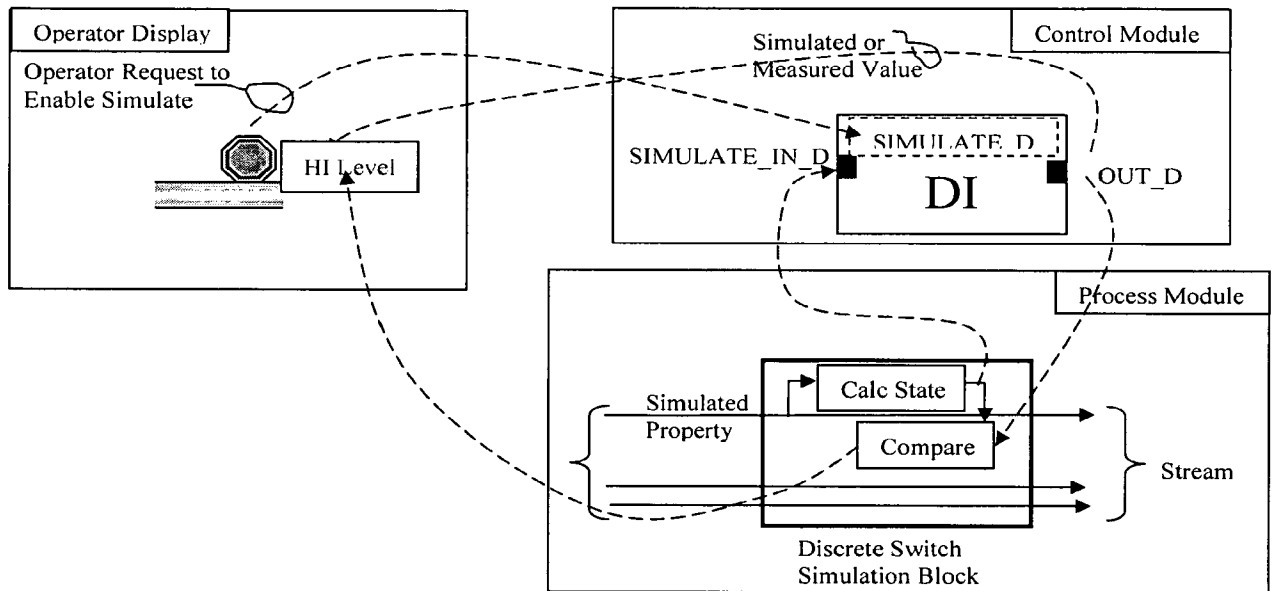
3.3.1.2.1 Transmitters

The field measurement value will normally be displayed by the transmitter element in the operator graphic. However, during operator training, the SIMULATE parameter of the associated AI in a control module will be set to Enable so that the simulated value may be used in the operator display and the associated control module. During normal process operation, the field measurement value will normally be displayed and used in the control module. If the associated transmitter fails, then the operator may elect to use the simulated value in the display and in the control. Also, the measured value will normally be used in the process simulation and automatically display and compared to the simulated value to the measured value, as illustrated below.



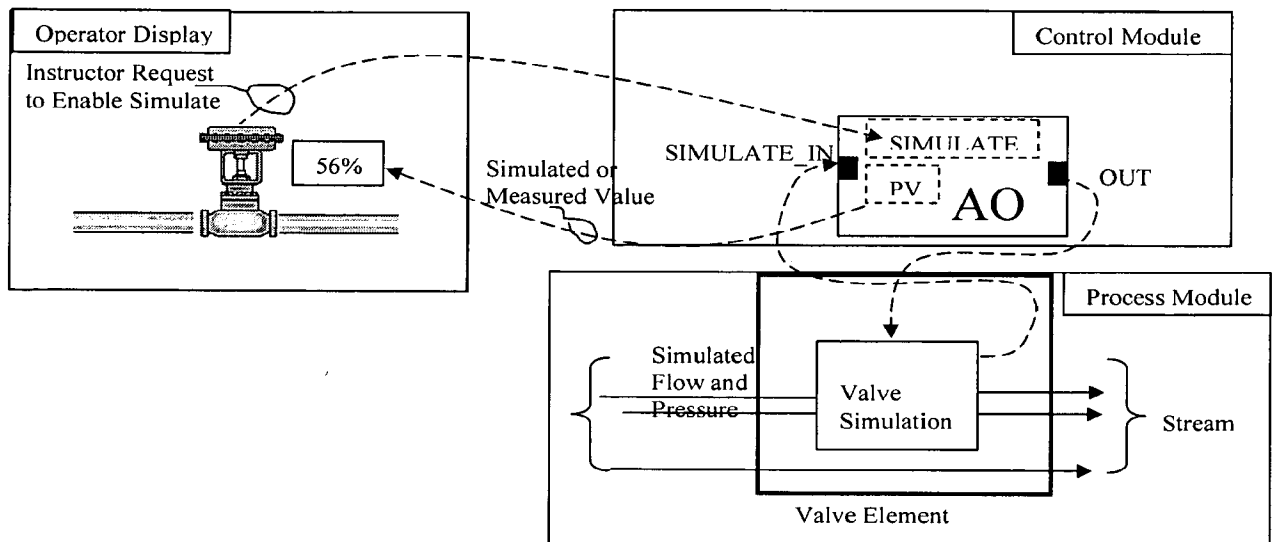
3.3.1.2.2 Switch Inputs

The field switch input will normally be displayed by the switch graphic element in the operator graphic. However, during operator training, the SIMULATE parameter of the associated DI block in the control module will be set to Enable so that the simulated value may be used in the operator display and the associated control module. During normal process operation, the field measurement value will normally be displayed and used in the control module. If the associated transmitter fails, then the operator may elect to use the simulated value in the display and in the control. Also, the measured value will normally be used in the process simulation and automatically display and compared to the simulated value to the measured value, as illustrated below.



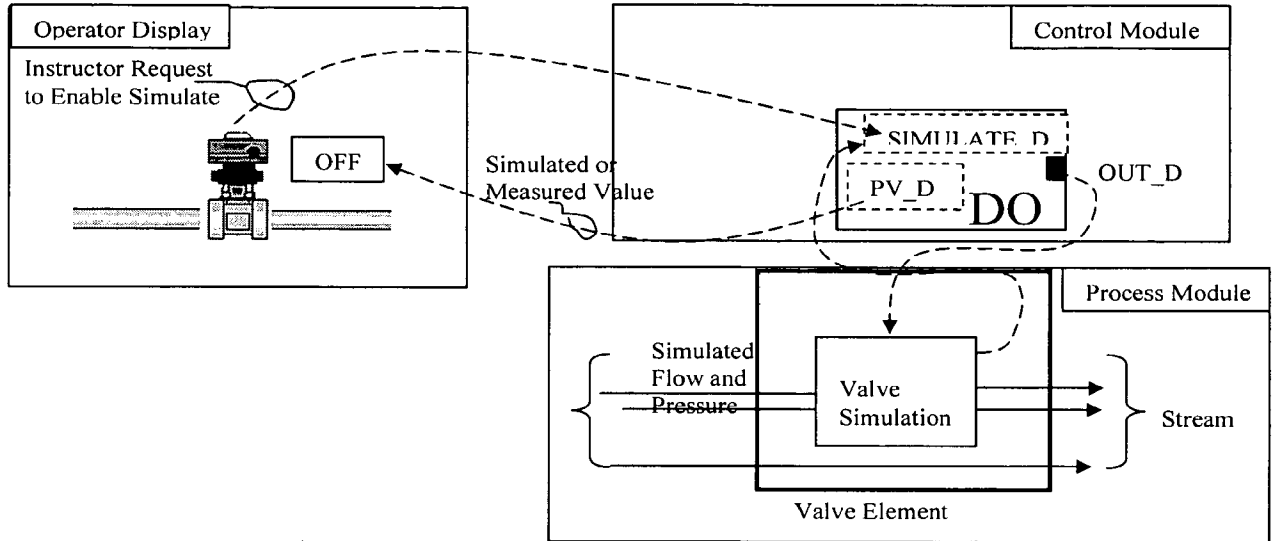
3.3.1.2.3 Regulating Valve

The measured stem position (for FF device) or the implied position of regulating valve will normally be displayed with the valve in the operator graphic. However, during operator training, the SIMULATE parameter may be Enabled so that the simulated value position may be used in the operator display and the associated control module. Through this simulation, it is possible to demonstrate the impact of valve stiction during maintenance and operator training, as illustrated below.



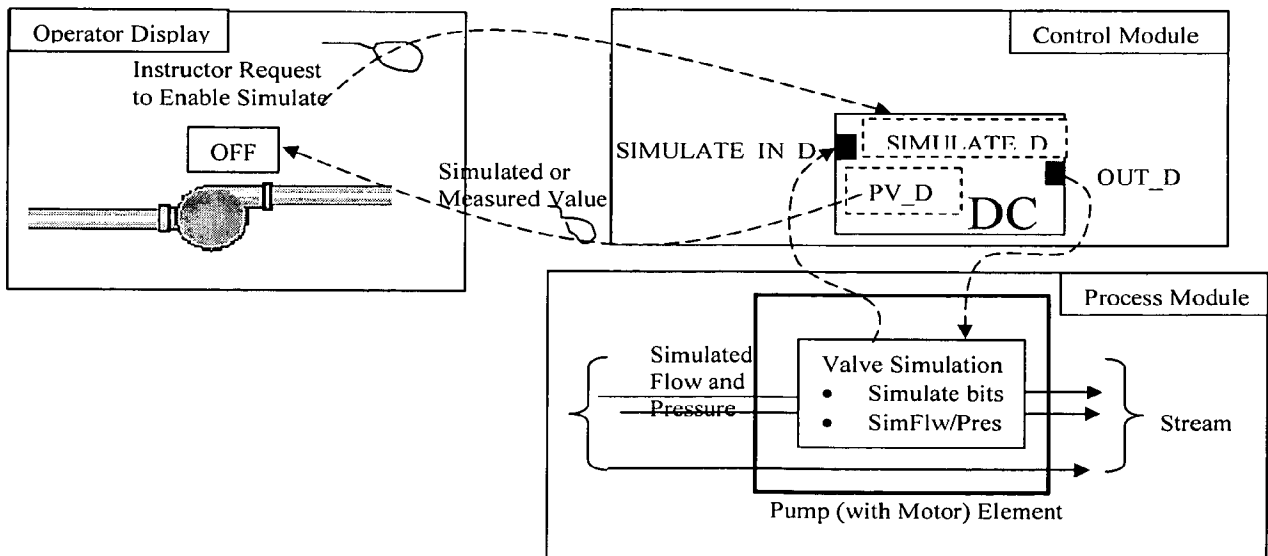
3.3.1.2.4 On-Off Valve

The limit switch feedback or the implied position of an On-Off valve will normally be displayed with the valve in the operator graphic. However, during operator training, the SIMULATE_D parameter may be Enabled so that the simulated On-Off position may be used in the operator display and the associated control module. Through this simulation, it is possible to demonstrate the impact of valve or actuator failure during maintenance and operator training, as illustrated below.



3.3.1.2.5 Pump(with Motor – fixed or multi-Speed)

The device control block may be used to provide the start/stop discrete outputs and discrete feedback from a pump motor actuator. The PV of the DC function block used for motor control may be shown in the graphic display to indicate the true status of the motor. However, during operator training or control checkout, the SIMULATE_D parameter may be Enabled so that the simulated On-Off status of the motor starter may be shown in the operator graphics and used in the DC function block. Through this simulation, it is possible to demonstrate the impact of a motor failure during maintenance and operator training, as illustrated below.



4 Scenario

4.1 Overview

This scenario is intended to illustrate the principles of the use of the combination of Process Graphics and Process Modules. In this scenario a plant has a number of heat exchanger units that are used to make Vinegar. Each Unit mixes a specified quantity of acid, base and water to produce a variety of vinegar. Each heat exchanger unit has as a feed acid, base, and water. Each heat exchanger has an outlet that goes to a product tank and a second outlet that recycles cooling water. The overall process is illustrated below.

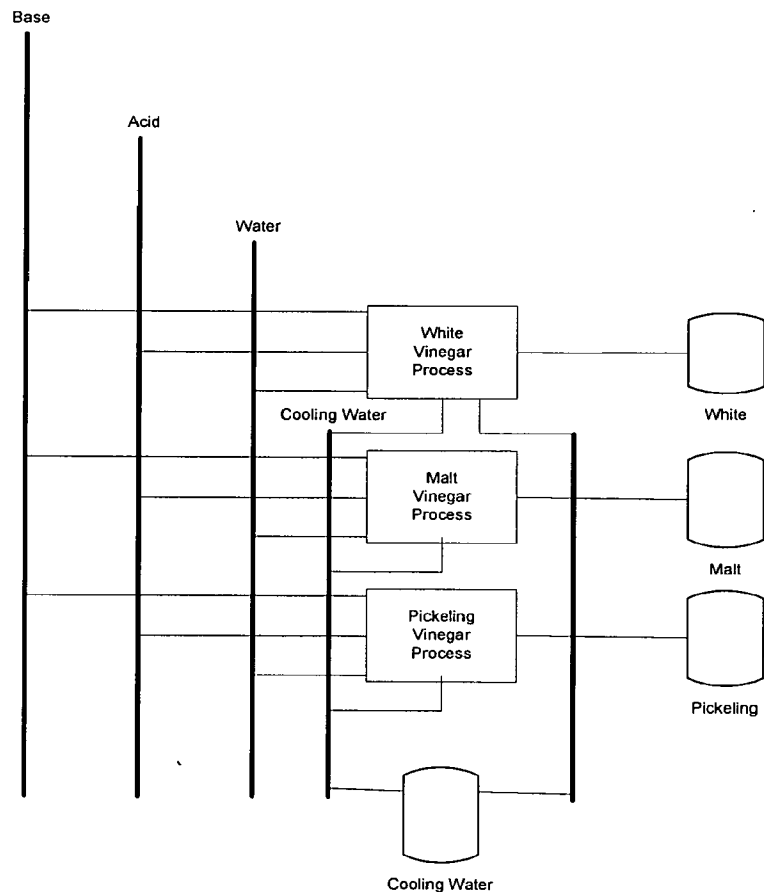


Figure 1. Vinegar Production

Each Unit has its own control strategy, process graphic, and process module. The scenario describes the development of a Process Graphic and a Process Module. The Process Graphic and Process Module can be used to provide Simulation capabilities for Control, Operator Training, etc.

To make the scenario easier to follow a possible control strategy is shown below. The drawing presents the control strategy from a Module Class point of view.

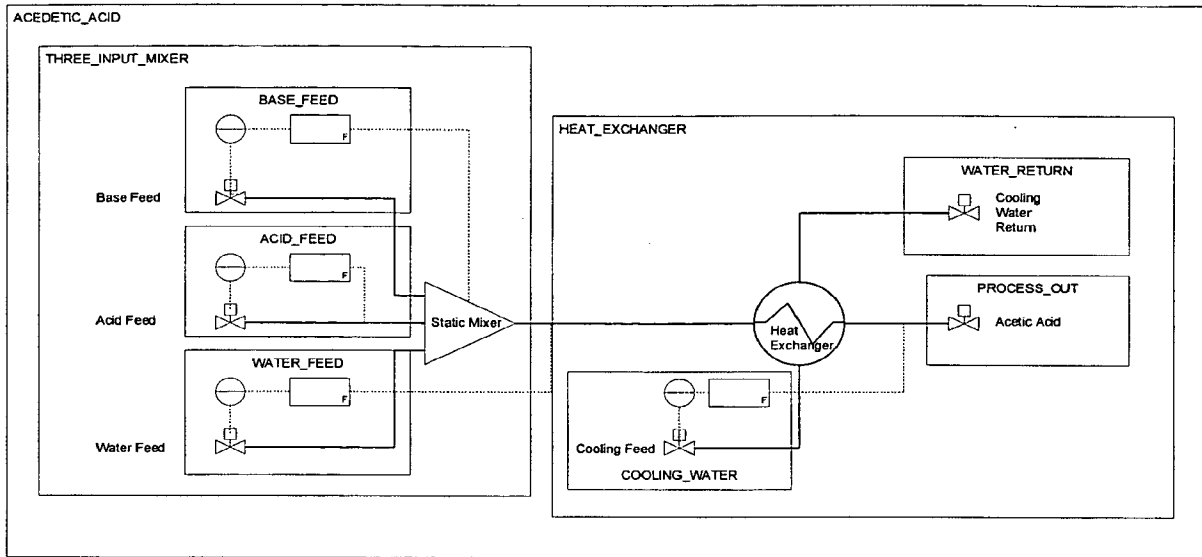


Figure 2. Vinegar Control Strategy

The Class-based hierarchy for the above drawing is shown below.

ACETIC_ACID	[Unit Class]
CIP	[Phase Class]
START	[Phase Class]
RUN	[Phase Class]
SHUTDOWN	[Phase Class]
THREE_INPUT_MIXER	[Equipment Class "STATIC_MIXER"]
BASE_FEED	[Module Class "FLOW_LOOP"]
ACID_FEED	[Module Class "FLOW_LOOP"]
WATER_FEED	[Module Class "FLOW_LOOP"]
HEAT_EXCHANGER	[Equipment Class "HEAT_EXCHANGER"]
COOLING_WATER	[Module Class "FLOW_LOOP"]
WATER_RETURN	[Module Class "ON_OFF_VALVE"]
PROCESS_OUT	[Module Class "ON_OFF_VALVE"]
NEUTRALIZER	[Module Class "PH_CONTROL"]

4.2 Process Graphic and Process Module Design

4.2.1 Process Graphic

This section discusses the design of Process Graphic Displays and Process Modules. To illustrate these ideas a Process Graphic Display is implemented covering a heat exchanger used for making vinegar.

The User creates their Process Graphic Display as much as they do with iFIX today. Essentially they create a Process Graphic Display and build it up using Process Graphic Items, Piping, etc.

The Process Graphic Display used in this scenario includes several ...

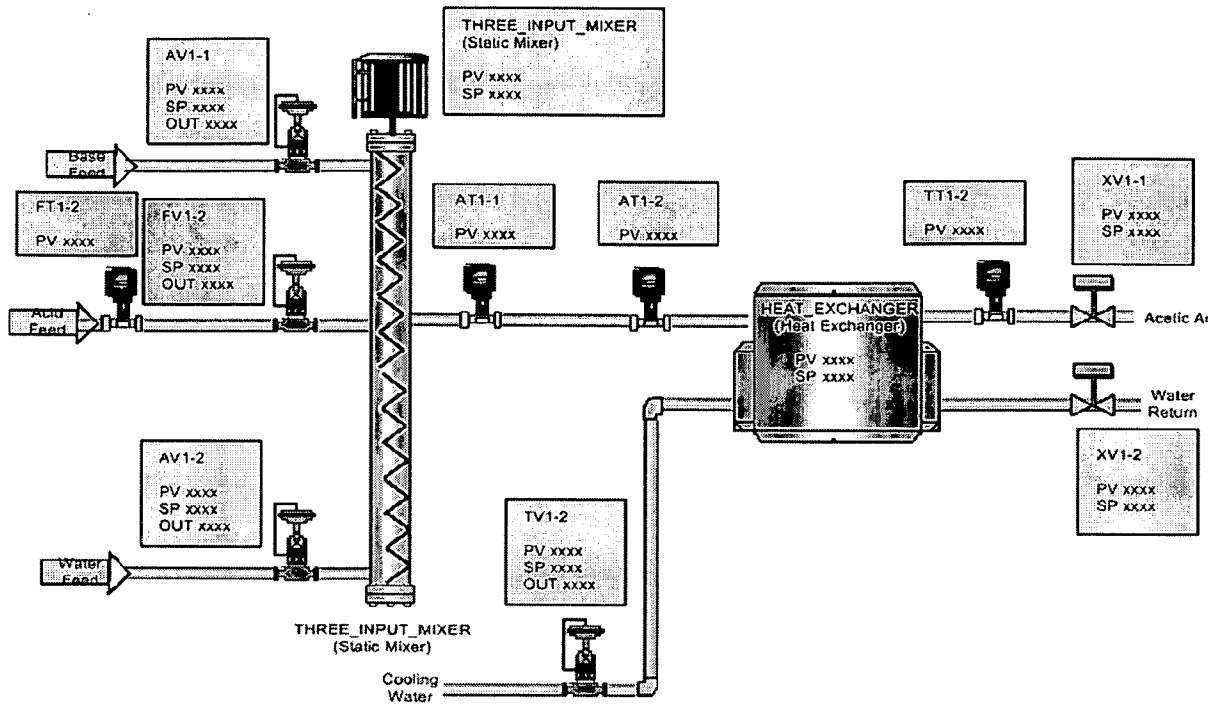


Figure 3. Process Graphic Display

4.2.2 Thumbnail View of the Process Graphic

Process Graphic Display that are created “Class-based” have a built-in capability – they can be collapsed (very much like composites when they are embedded in other displays). This is illustrated below for the Vinegar process.

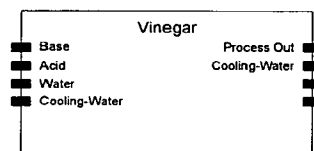


Figure 4. Process Graphic Display Composite

Displays can then be combined to create a large navigational display. This is shown for the overall Vinegar example described in this scenario.

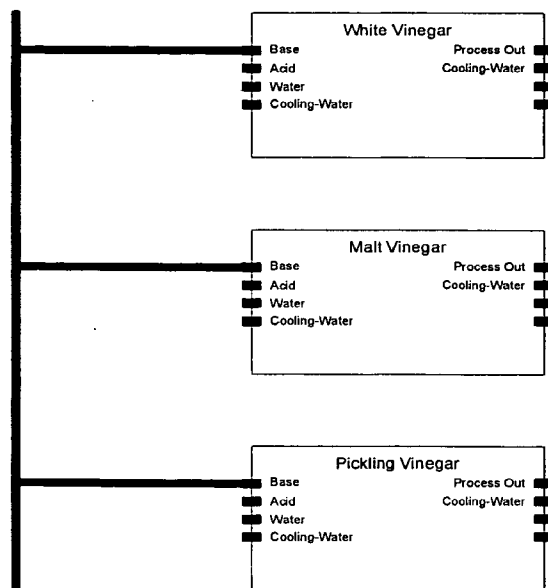


Figure 5. Overview Process Graphic Displays

These overview displays can then be used for navigation.

4.2.3 Process Module

4.2.3.1 Process Module View

The Process Module View includes...

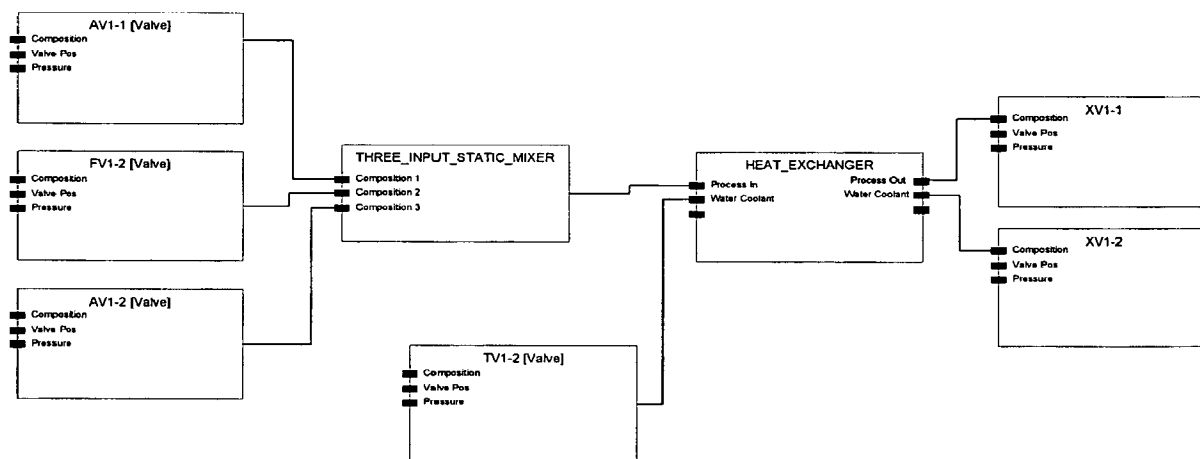


Figure 6. Process Module Display

4.2.3.2 Energy Balance

4.2.3.3 Mass Balance

4.2.3.4 Simple Composition

4.2.3.5 Custom Composition

4.3 Process Simulation Using HYSYS

The Process Simulation Blocks described above can be given high fidelity behavior by tying them into a simulation packages such as HYSYS from Hyprotech. The following part of the scenario substitutes HSYS in to provide behavior on the Simulation Function Blocks.

4.3.1 Dynamic Simulation

When the user creates their Process Module they are able to specify that a High Fidelity Simulation will be used. When they do this the built-in behaviors in the Process Module will be disabled. In this case the Process Module acts as a 'composite' to marshal parameters to/from the High Fidelity Simulation System.

In the scenario that is being described here the Acetic Acid process is reproduced in HYSYS. The simulation is shown below.

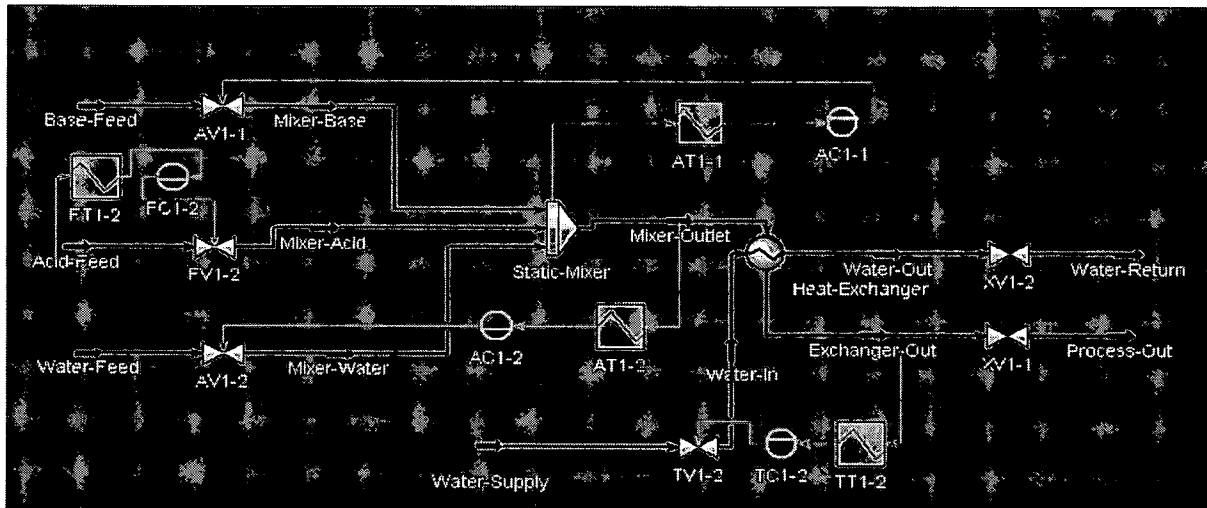


Figure 7. Acetic Acid High Fidelity Simulation

The inputs and outputs for this simulation are shown in the attached figure.

HYSYS DCS Interface

Exports

Index	PV Object	PV Var	Status	Value	Units	Multiplier	Bias	Machine	Server	Item ID
0	AT1-1	OP Value	DCS	7.983		1.000	0.0000	usaust-deliber	OPC.DeltaV.1	FIG9_50/AT1-1/SIMULATE_IN.CV
1	AT1-2	OP Value	DCS	55.00	lb/ft3	1.000	0.0000	usaust-deliber	OPC.DeltaV.1	FIG9_50/AT1-2/SIMULATE_IN.CV
2	TV1-2	Percentage o	DCS	0.00	%	1.000	0.0000	usaust-deliber	OPC.DeltaV.1	FIG9_50/TY1-2/SIMULATE_IN.CV
3	FT1-2	OP Value	DCS	3.098e-013	kpph*	1.000	0.0000	usaust-deliber	OPC.DeltaV.1	FIG9_50/FT1-2/SIMULATE_IN.CV

Edit

Insert

Delete

Drivers

General Data

Controllers

PV Export

PV Import

Clear

Stop Integrator

Disable

DCS Interface

	PV Object	PV Var	Status	Value	Units	Multiplier	Bias	Machine	Server	Item ID	InitItem ID	ModelInit Item ID
0	AV1-1	Actuator Desi	DCS	9.45	%	1.000	0.0000	usaust-deliber	OPC.DeltaV.1	FIG9_50/AV1-1/OUT.CV	FIG9_50/AV1-1/SP.CV	FIG9_50/AV1-1/MODE.T
1	AV1-2	Actuator Desi	DCS	0.00	%	1.000	0.0000	usaust-deliber	OPC.DeltaV.1	FIG9_50/AV1-2/OUT.CV	FIG9_50/AV1-2/SP.CV	FIG9_50/AV1-2/MODE.T
2	FV1-2	Actuator Desi	DCS	0.00	%	1.000	0.0000	usaust-deliber	OPC.DeltaV.1	FIG9_50/FV1-2/OUT.CV	FIG9_50/FV1-2/SP.CV	FIG9_50/FV1-2/MODE.T

Insert

Delete

General Data

Controllers

PV Export

PV Import

Stop Integrator

Disable

Figure 8 High Fidelity Simulation Import/Export Mappings

Process Modules Simulation

Table of Contents:

1	INTRODUCTION.....	555
1.1	BACKGROUND – FLUID AND GAS FLOW SIMULATION.....	555
1.1.1	Specified Starting Pressure	555
1.1.2	Vessel Establishes Starting Pressure.....	556
1.1.3	Pump or Fan Establishes Starting Pressure.....	556
2	TECHNIQUES FOR SIMULATION.....	557
2.1	LOW FIDELITY PROCESS SIMULATION	558
2.2	MEDIUM AND HIGH FIDELITY MODELING TECHNIQUE	559
2.2.1	Sequential-Modular Simulation.....	559
2.2.2	Open Equation /Simultaneous Simulation	560
2.2.3	Parameter Estimation and Data Reconciliation	560
2.3	STREAM AND ELEMENT PROPERTIES	561
2.3.1	Parameter Engineering Units.....	561
2.3.2	Stream Composition	563
3	SIMULATION USING DELTAV PROCESS MODULES	566
3.1	STREAM ELEMENT.....	567
3.2	REGULATING VALVE	568
3.3	BLOCKING VALVE	569
3.4	FIXED SPEED CENTRIFUGAL PUMP	569
3.5	PIPE SEGMENT.....	570
3.6	HEAT EXCHANGER.....	572
3.7	VESSEL SIMULATION.....	573
3.7.1	Atmospheric Tank	573
4	EXAMPLE IMPLEMENTATION.....	573
4.1	NAMED SET DEFINITIONS.....	574
4.2	NAMED BASIC PROPERTIES – USE OF ARRAY PARAMETER.....	575
4.3	NAMED STEAM ELEMENT	577
4.3.1	Stream Parameters.....	577
4.3.2	Stream Algorithm	577
4.4	CONSTANT SPEED CENTRIFUGAL PUMP ELEMENT	578
4.4.1	Pump Parameters	578
4.4.2	Pump Algorithm	578
4.5	PIPE ELEMENT	578
4.5.1	Pipe Parameters	579
4.5.2	Pipe Algorithm.....	579
4.6	REGULATING VALVE ELEMENT	580
4.6.1	Regulating Valve Parameters	580
4.6.2	Regulating Valve Algorithm.....	581
4.7	BLOCKING VALVE ELEMENT	582
4.7.1	Blocking Valve Parameters.....	582
4.7.2	Blocking Valve Algorithm	582
4.8	CONVERGENCE ELEMENT	582
4.8.1	Convergence Parameters	583
4.8.2	Convergence Algorithm	583
4.9	TEST VESSEL ELEMENT	584
4.9.1	Test Vessel Parameters.....	584
4.9.2	Test Vessel Algorithm	584
4.10	TANK ELEMENT.....	585
4.10.1	Tank Parameters	585

4.10.2	Tank Algorithm	586
4.11	HEAT EXCHANGE ELEMENT	587
4.11.1	Heat Exchanger Parameters	587
4.11.2	Heat Exchange Algorithm – OutletTemperature and Heat Tranfer	587
4.11.3	Heat Exchange Algorithm – Flow and Pressure for Input 1.....	588
4.11.4	Heat Exchange Algorithm – Flow and Pressure for Input 2.....	588
4.12	VARIABLE SPEED PUMP ELEMENT	590
4.12.1	Variable Speed Pump Parameters	590
4.12.2	Variable Speed Pump Algorithm.....	591
5	APPENDIX – CONVERSION TABLE.....	592
5.1	AREA	592
5.2	ENERGY (INCLUDES WORK)	592
5.3	FORCE	593
5.4	HEAT (AVAILABLE ENERGY)	593
5.5	COEFFICIENT OF HEAT TRANSFER.....	593
5.6	DENSITY OF HEAT FLOW RATE.....	594
5.7	HEAT CAPACITY AND ENTROPY	594
5.8	HEAT FLOW RATE.....	594
5.9	SPECIFIC HEAT CAPACITY AND SPECIFIC ENTROPY	595
5.10	THERMAL CONDUCTIVITY	595
5.11	THERMAL INSULANCE	595
5.12	THERMAL RESISTANCE.....	595
5.13	THERMAL RESISTIVITY	596
5.14	LENGTH.....	596
5.15	MASS	596
5.16	MASS DIVIDED BY TIME (INCLUDES FLOW).....	597
5.17	MASS DIVIDED BY VOLUME (INCLUDES MASS DENSITY AND CONCENTRATION).....	597
5.18	POWER	597
5.19	PRESSURE OR STRESS (FORCE DIVIDED BY AREA).....	598
5.20	TEMPERATURE	599
5.21	TEMPERATURE INTERVAL	599
5.22	TIME	599
5.23	VELOCITY (INCLUDES SPEED).....	600
5.24	VISCOSITY, DYNAMIC	600
5.25	VISCOSITY, KINEMATIC.....	600
5.26	VOLUME (INCLUDES CAPACITY)	601
5.27	VOLUME DIVIDED BY TIME (INCLUDES FLOW).....	601

1 Introduction

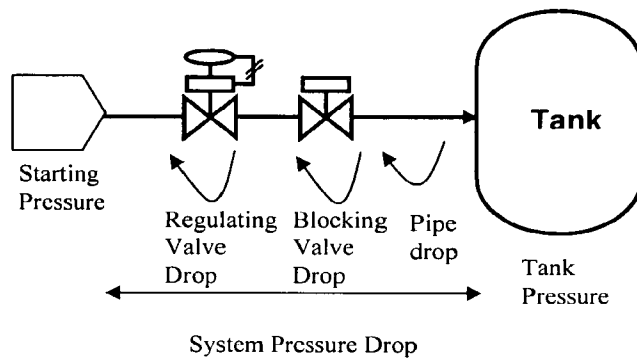
The implementation of process simulation is a significant part of the project. Many of the basic concepts associated with the creation of simulation based on the operator graphic display have been defined in the Starburn concept document. Also, this concept document defined the process module as the container for process element used to construct the simulation. Some of the most common components, such as stream elements, process elements that simulate actuators, and processing units for vessels such as tanks are defined. The purpose of this document is to document the difference approaches that can be taken in the implementation of process simulation. Also, an approach for the DeltaV simulation implementation is presented and details provided on some basic stream components and process vessel. In the first two sections, some background is provided on process simulation and traditional approaches in implementation. The later sections focus on the implementation approach that will be taken in the DeltaV Process Module.

1.1 Background – Fluid and Gas Flow Simulation

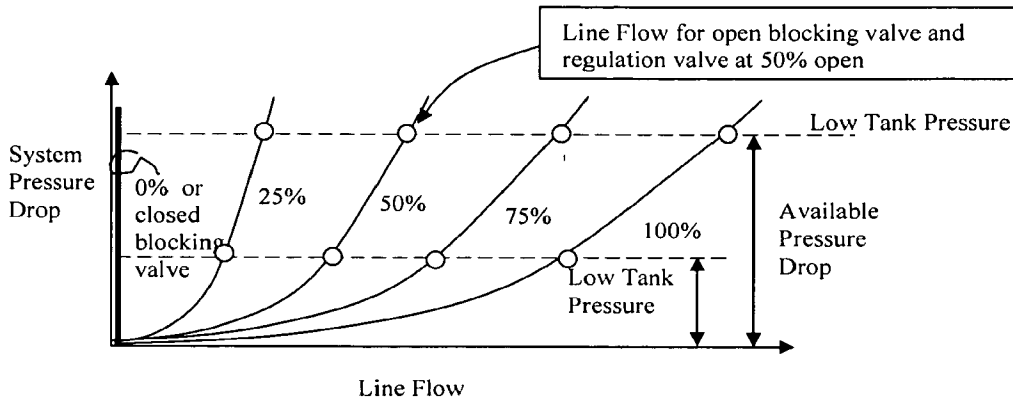
Pressure is driving force for liquids or gas transfer between vessels. The flow through a transfer pipe or duct is based on the starting pressure and the pressure drops introduced by the processing elements in the flow path e.g. pipe, valves and the point of discharge. To illustrate this, in this section we will consider three typical examples that will be encountered in process simulation

1.1.1 Specified Starting Pressure

In some cases, the upstream equipment may not be included in the simulation and the user may specify the starting pressure as part of the stream properties when pressure is defined as the independent parameter. When steam pressure (starting pressure) is specified, then the flow through the associated pipe is determined the pressure drop introduced by the pipe and the elements in the pipe and the pressure at the point of discharge into a tank, as illustrated below.



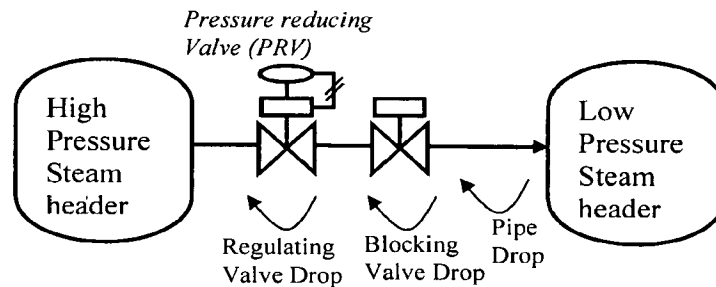
The calculation of the pipe flow is complicated by the fact that the pressure drop introduced by the pipe and processing elements in the pipe is a function of flow rate. Also, for regulation and on-off elements, a variable pressure drop is introduced based on the actuator position. Assuming the tank pressure may vary i.e. closed pressurized tank, then variation of pipe flow as a function of tank pressure, actuator position and characteristics can be illustrated as below.



Pipe Flow as a Function of System Pressure Drop and Available Pressure

1.1.2 Vessel Establishes Starting Pressure

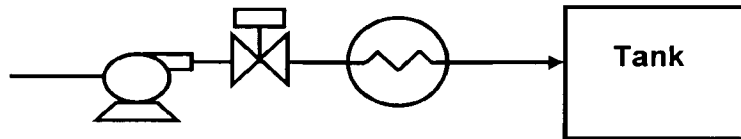
If a vessel is pressurized, then this pressure may be used as the driving force for transfer of vapor to liquids between tanks or from a tank to a vent, header, etc.



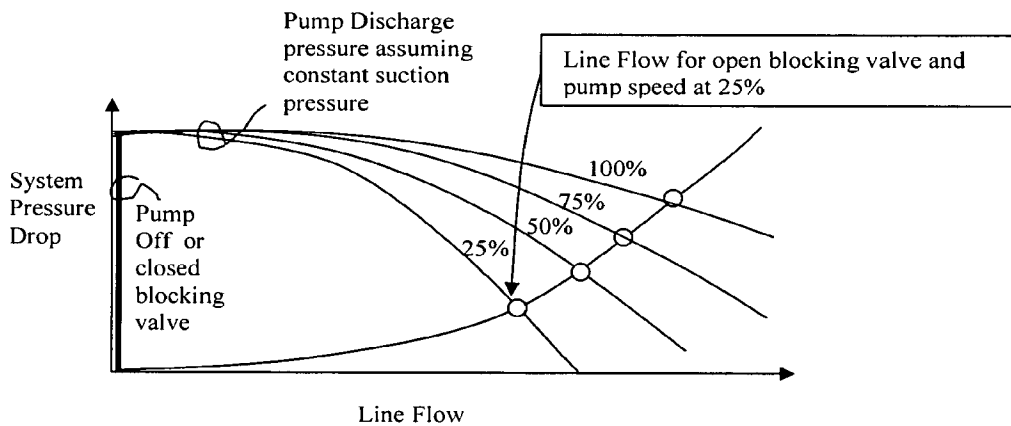
For this example, the driving force for steam flow is the differential pressure between the high and low steam header. Since the pressure in each header, then the differential pressure is variable. The pipe flow thus is determined by this differential pressure, the position of the regulating and on-off valve, and line loss. This relationship is similar to that previously illustrated for the case where the starting pressure was specified.

1.1.3 Pump or Fan Establishes Starting Pressure

A pump or fan is often the pressure source that provides the driving force for liquid or gas flow respectively. In such cases, the pressure provide will be a function of the flow. This relationship may be documented by the equipment manufacturer by a pump or fan curve that shows the expected pump differential pressure as a function of pump flow. The following variable speed pump example may be used to illustrate the impact of pump discharge pressure with flow rate. In this example, the liquid supply to the pump is assumed to be from a tank at constant (atmospheric) pressure. In this example, pump discharge flow travels through a pipe to a heat exchanger before entering another tank (at atmospheric pressure) as illustrated below.



Assuming the downstream tank pressure is constant i.e. open tank, then variation of pipe flow as a function of pump speed, actuator position and characteristics and pipe pressure loss can be illustrated as below.



Pipe Flow as a Function of System Pressure Drop and Available Pump Pressure

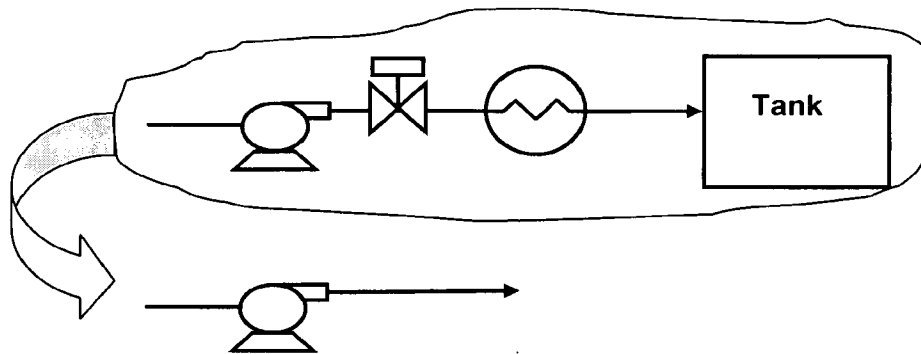
At a given pump speed, a flow will be established at which the discharge pressure matches the system pressure loss. Since the system pressure loss is a function of the flow rate, then a point of equilibrium may be established. The flow will change with any variation in system pressure loss, downstream tank pressure or changes in pump suction pressure.

2 Techniques for Simulation

Various approaches may be taken in process simulation. In this section we review traditional techniques that have been successfully used to implement low fidelity and medium to high fidelity process simulation.

2.1 Low Fidelity Process Simulation

If a process simulation is to capture the true interdependence of pressure and flow in a stream, then the solution is non-trivial. Thus, in many cases low fidelity process simulation packages have been implemented for operator training systems and for control or graphics checkout. These low fidelity process simulation systems do not attempt to model all the process behavior. For example, the flow in a pipe may be determined strictly by the position of the regulating actuator in the pipe. The impact of upstream or downstream pressure, pressure losses through the pipe loss, blocking valves, or inline elements such as heat exchangers are often **totally ignored**. Thus, for the calculation of stream flow, the process design may be simplified for as follows for a low fidelity simulation.



For example, the flow associated with a variable speed pump may be assumed to be linear with pump speed – allowing the flow to be calculated simply as:

$$\text{Stream Flow} = (\% \text{ Pump Speed}/100) * K$$

Where K is a constant based on the EU range of an associated flow transmitter

Though such an approach leads to a simulation that is quick and easy to implement, this implementation has the following disadvantages in actual practice.

- The pressures in vessels connected by a stream have no impact on the calculated stream flow and pressure. For example, a flow may be calculated even though the pressure driving force is zero.
- Processing elements in the stream have no impact on the calculated pressure and flow. Thus, a flow may be calculated even when block valve(s) upstream or downstream of the regulating valve or pump are closed.
- Characteristics of the regulating elements and the variations in system loss with flow are not reflected in the simulation. Thus, the calculated stream flows pressure at different points along the stream may have significant error.

To address some of these shortcomings, the vendor of low fidelity process simulation packages allow the user to introduce heuristics based on the specific processing element and vessels associated with the stream. For example, to account for the fact that a block valve is upstream of a regulating valve, the user would have to add logic that does the equivalent of the following:

```
If (Block Valve HC101) = Closed Then
    Calculated Flow = 0
Endif
```

The addition of such heuristics to account for these conditions are avoided in many medium and high fidelity process simulation by accurately modeling the stream pressure source, the impact of each element in the process stream and the downstream vessel pressure. Also, since these components are accounted for in simulation, then the stream flow and pressure conditions along the way may be more precisely calculated over a wide range of operating conditions. Thus, it is proposed that the stream modeling capability provided by the DeltaV process module and processing elements be designed to allow the impact of each element in the stream to be accurately modeled.

2.2 Medium and High Fidelity Modeling Technique

Commercial products that support complete dynamic process simulation are typically based on one of two basic approaches:

Sequential-Modular Simulation – Classical approach is utilized in the majority of simulation tools on the market today

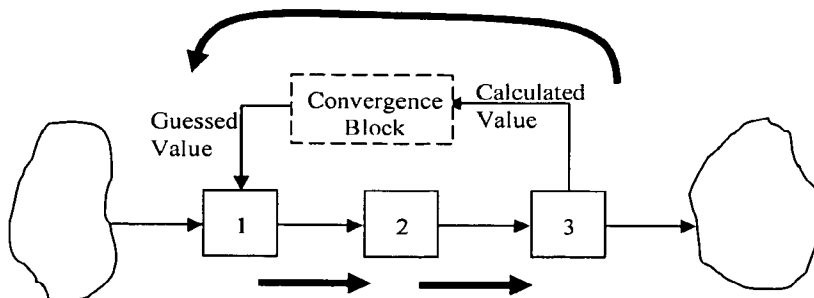
Open Equation Simulation – Latest approach that is promoted by some vendors for simulation of larger processes.

Using either approach, there is a need to tune model parameters to reflect the actual operational behavior. In this section, we compare these approaches and techniques for model adjustment and data reconciliation.

2.2.1 Sequential-Modular Simulation

With this approach, each piece of equipment/unit is represented as a “module” that is characterized by input/output connections and a contained algorithm. Each equipment/unit module is solved one after another, sequentially. Because of the interactions between modules, the sequential model requires several iterations in each simulation step in order to achieve convergence. To speed convergence, some products are structured for *simultaneous solved pressure flow networks*, with the remainder of the model solved in a sequential-modular fashion.

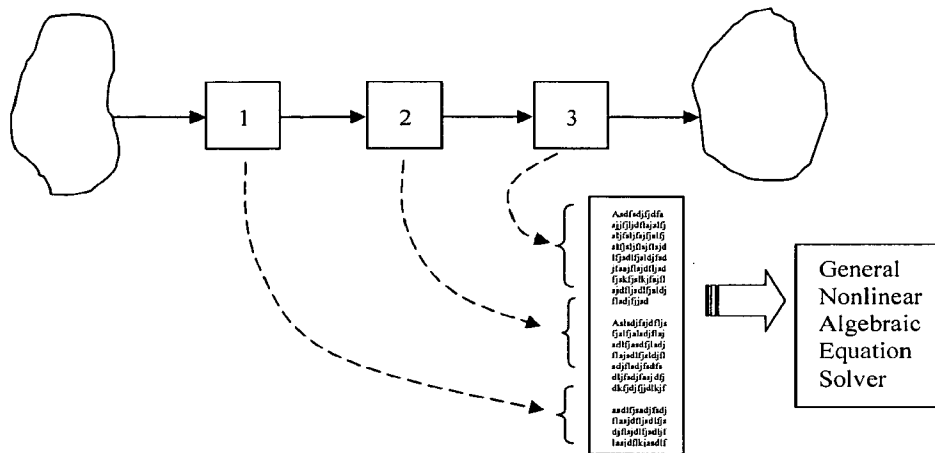
To utilize the sequential-modular approach when the process is characterized by interactions such as flow pressure within a stream, the concept of “tearing” connections to modules may be introduced. An initial guess is provided for values that are unknown (the tear variables) and then calculate the value of the tear variables from the modules. These calculated values may be used to determine new guesses that are applied in an interactive manner until the difference between the estimated and calculated values are sufficiently small. This process can be represented in the following manner (reference: Edgar, Optimization in Large-Scale Plant Design and Operations):



One of the strengths of the sequential –modular approach is that the representation of each piece of equipment can be standardized and combined in a variety of ways to simulate different equipment configurations.

2.2.2 Open Equation /Simultaneous Simulation

This latest technique is based on simultaneous, equation based solutions technology. The differential and algebraic equations from all the contained model objects are treated as a single large set of equations. These equations are solved simultaneously. This general approach applies to all kinds of simulation and can be conceptually represented as follows.



A good deal of complexity is introduced by this approach to successfully address the different combinations of process elements and associated discontinuities.

The EU-sponsored CAPE-OPEN committee is defining interface standard for process simulation tools. The current specification requires the modeler to provide access to the equation structure and initial values. Such information allows an open equation approach is simulation implementation. However, addressing discontinuities, such as on-off valves, in such an environment is a significant challenge and has limited the success of products following this standard.

2.2.3 Parameter Estimation and Data Reconciliation

During model creation, a number of parameters may be specified that characterize the equipment design and capability e.g. heat-exchanger coefficients, etc. Since the actual equipment performance may not match that assumed in the design, it is necessary to tune these parameters to make the model match the actual process. In some cases, the process simulation supplier offers tools to allow this to be done on-line. For example, the ABB Dynamic Solutions package offers a “parameter estimation capability. Uncertain or unmeasured parameters can automatically be estimated from operational data. The estimated parameters are optimal in the sense that they minimize the quadratic cost of model mismatch from actual behavior. In this way, the process model can be kept updated to reflect the plant operation.

To account for measurement drift and measurement error or fault, many simulation packages incorporate an ability to do data reconciliation. For example, the ABB Dynamic Solutions, the process model is used to correct measurements are "cleaned" for gross errors and then the model is used to calculate bias values for measurement values to "punish inconsistencies as well as corrections". This technique is also used to rectify material and energy balances and to provide lead detection capability.

2.3 Stream and Element Properties

The standard properties that are required for liquid, gas, and solids connections in the forward path are:

- 1) Mass-flow
- 2) Temperature
- 3) Pressure
- 4) Density
- 5) Specific Heat.

These properties are required to do the energy-material balances associated with a process element or vessel such as a tank or mixer.

For example, the tank algorithm would calculate the level based on the inputs connection mass-flow, outlet connections mass-flow, inlet density (to convert mass flow to volume) and factors that characterize the physical shape of the tank i.e. liters/meter, height of tank. Based on the level of the tank, the outlet connection pressure would be calculated based on overhead tank pressure (if pressurized) and the outlet connection points in the tank, and the level plus density (to determine liquid head). Similarly, the outlet connection temperature will be calculated based on the inlet connection temperatures, mass flow and specific heat, tanks level (and specific heat and density).

In this section we address the units associated with a stream or element property. Also, ability to associate composition with each independent stream is defined along with the rule for propagation of composition through processing elements and vessels.

2.3.1 Parameter Engineering Units

All properties associated with a connection, processing element or vessel are required to be in the SI units. The standard properties will always be internally calculated and exchanged between vessels as streams in metric units - Based on the International System of Units (SI) and SI Derived Units that are commonly used in specialized fields such as process measurement and control. This selection of units is based on Federal Standard 376B, Preferred Metric Units for General Use by the Federal Government:

Parameter	Unit Name	Unit Symbol	Express in other SI Units
Mass-flow	kilogram per second	kg/s	
Temperature	degree Celsius	°C	K
Pressure	kilopascal	kPa	N/m ³
Density	kilogram per cubic meter	kg/m ³	
Specific Heat	kilojoule per kilogram kelvin	kJ/(kg·K)	

Other contained or internal properties may be associated with vessels or processing elements. The following units should be used for visible or internal properties:

Parameter	Unit Name	Unit Symbol	Express in other SI Units
Length	meter	m	
Area	square meter	m ²	
Volume	cubic meter	m ³	
	liter	L	m ³
Time	second	s	
Velocity	meter per second	m/s	
Acceleration	meter per second squared	m/s ²	
Flow rate	cubic meters per second	m ³ /s	
	liters per second	L/s	m ³ /s
Viscosity(dynamic)	millipascal second	mPa·s	
Force	newton	N	kg·m/s ²
Energy	kilojoule	kJ	N·m
Power	kilowatt	kW	N·m/s
Thermal conductivity	watt per meter kelvin	W/(m·K)	
Coefficient of heat transfer	watt per square meter	W/(m ² ·K)	

The following common metric prefixes may be attached to the SI unit name to form multiples of the SI unit.

Prefix Name	Prefix Symbol	Multiplication factor
kilo	k	10 ³

The user may select whether to enter and see process block parameters in metric units or Inch-pound units. Based on this selection, the following will occur:

- The default parameters of function blocks will be set to either Metric or Inch-pound units
- The process function block calculation will take into account the SI or inch-pound unit selection

The set of default parameter values and internal process block calculations will be based on the following multiplication factor used to convert from inch-pound units to Metric.

Measure	To Convert From	To	Multiply By
Mass-flow	Pounds per second	kilogram per second	0.45359237
Temperature	degrees Fahrenheit	degree Celsius	(t _F -32)/1.8
Pressure	Pound-force per square inch	kilopascal	6.894757
Density	Pounds per cubic foot	kilogram per cubic meter	16.01846
Specific Heat	Btu per pound degree Fahrenheit	kilojoule per kilogram Kelvin *	4.1868

Length	foot	meter	0.3048
Area	square foot	square meter	0.09290304
Volume	Cubic foot	cubic meter	0.02831685
	gallon	liter	3.785412
Time	second	second	1
Velocity	Foot per second	meter per second	0.3048
Acceleration	Foot per second squared	meter per second squared	0.3048
Flow rate	Cubic foot per second	Cubic meters per second	0.02831685
	Gallon per second	Liters per second	3.785412
Viscosity(dynamic)	centipoise	millipascal second	1
Force	Pound-force	newton	4.448222
Energy	Btu	kilojoule	1.055056
Power	Btu per second	kilowatt	1.055056
Thermal conductivity	Btu inch per hour square foot degree Fahrenheit	watt per meter Kelvin*	0.1442279
Coefficient of heat transfer	Btu per hour square foot degree Fahrenheit	watt per square meter Kelvin*	5.678263

* Defined in terms of temperature interval. Therefore K may be replaced by °C.

2.3.2 Stream Composition

The specification of stream composition is optional in the process simulation i.e. may not be used/specified. Therefore, the user should be able to specify the connection type (gas, liquid, solid, energy) independent of the composition (air, water, fuel oil, etc). For energy connection types (i.e. current flow through a wire) there is no associated composition. All streams, processing elements, and standard vessels will be designed to process streams which have composition defined. Composition will be defined in terms of the mass fraction represented by each possible path component

The elements in a stream such as a valve, pipe, pump will have no impact on the stream composition i.e. the inlet and outlet composition are the same. Those elements that mix multiple inlet streams will use/change composition i.e. header or vessel. For these types of elements, the outlet stream may in general contain some concentration of the components that are associated with each inlet connections plus new components (if reaction occurs). Thus, when composition is defined for connections, then the same composition is always used for upstream and downstream connections to the element. In general, the same composition may be used for all elements in the same "independent stream path".

For example, in a boiler application, there are two "independent paths" through the process where a consistent composition should be defined/used by all elements in the path 1) water/steam path and 2) fuel and air/flue gas path. The water flowing through the feed water tank, deairator and steam in the drum, desuperheater, and steam header would all use the water/steam composition. Since the natural gas, oil, coal streams mix with the air flow stream to produce stack gas, then the same composition would apply to the fuel, air, and stack gas streams.

When a user defines a stream, he may elect to define the composition. When he chooses this option, then it will be necessary for him to define the stream composition by first identifying all the possible chemical elements that may be present in this stream as it is processed and mixed in the downstream equipment associated with the stream. Then he may define the composition of the stream in terms of the mass fraction of the elements that are present. In general, the composition would be defined as follows:

Composition Name

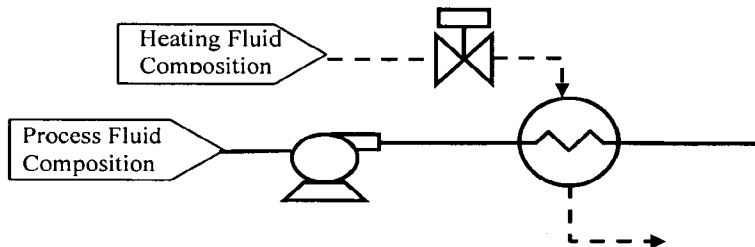
Possible Components	Mass Fraction
Component 1	0
Component 2	0
Component 3	0.2
Component 4	0.2
o	
o	
Component N	0.6

At any point in the processing of the stream, the sum of the mass fraction of the component in the stream may add up to one. The density property of the stream will be automatically calculated based on the stream composition.

In defining the stream composition, the user may select a predefined composition e.g. fuel or create a composition by defining components that may be present in the stream as it is processed. If components will be present that are not defined in the standard list component, then the ability is provided for the user to add components by defining:

- Component name
- Density
- Heat capacity

In a few cases, a processing element may be associated with more than one "independent path". For example, a heat exchanger has two stream inputs (steam, liquid stream) but these do not mix in the exchange. Thus, the composition on the steam input connection and condensate out connection would be the same composition i.e. are in the same "path", as illustrated below.



A processing element may be associated with more than one "independent path"

The composition of the exchanger liquid input connection and outlet connection may be the same but could be different than the steam path composition. Elements with multiple inputs in which mixing occurs e.g. tank, reactor, column, etc should require that the same composition (if defined) be used on all inputs and outputs since all inputs to this element are by definition in the same processing "path" as the outlet connections.

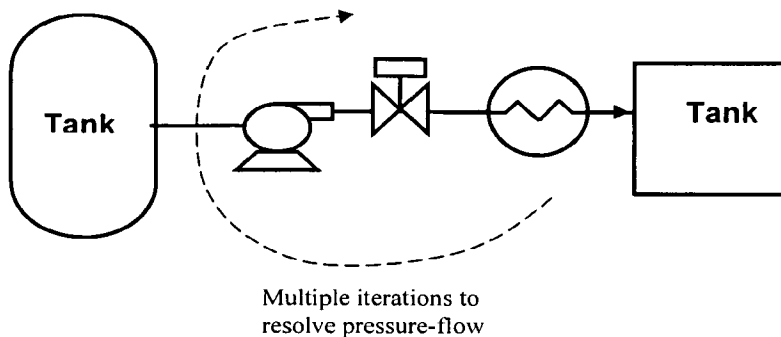
Based on this concept of "independent paths" through a process, we can enforce/require that the same composition (if specified by the user) be used for all elements associated with each "independent path" through the process. In some cases, there may be two or three "independent paths" through a process. If separate, distinct processing is done in different plant areas e.g. water treatment and pulping, then it may be necessary to define a composition for each area of the plant. In most cases the user will need to define two or three composition definition and these will be sufficient to address the entire plant simulation.

The stream composition and density property will be automatically propagate by stream elements that have a single input. However, any vessel with two or more input streams that are mixed in the vessel may calculate the resulting composition and associated density of the vessel outlet streams. For a simple tank in which the streams do not react, then the outlet composition may be calculated based on the inventory in the tank and the inlet stream flows along with the composition associated with inventory and streams.

The standard process blocks that will be provided for vessel simulation will assume that no reaction occurs in the vessel. The user may define a custom process block in which he takes into account reactions that may occur within the vessel. In these cases, the reaction kinetics will be defined in the algorithm implemented by the user.

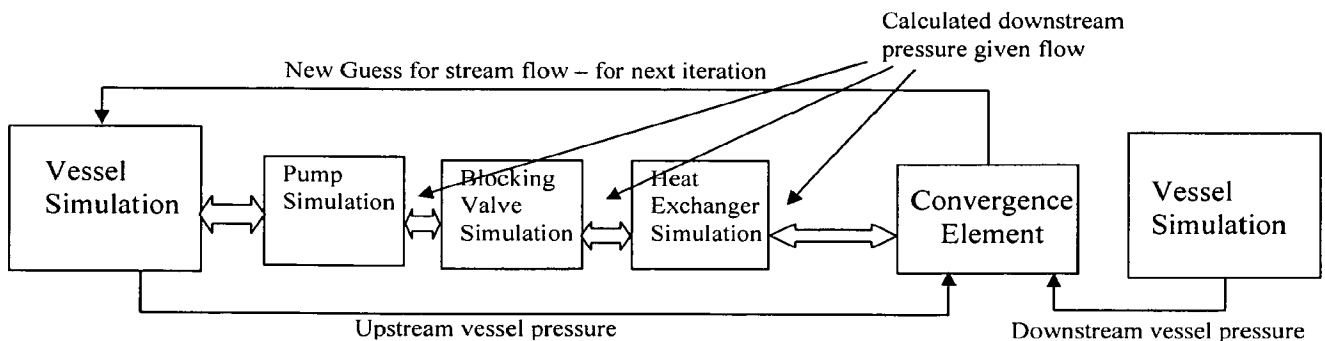
3 Simulation Using DeltaV Process Modules

The accurate calculation of the flows into and out of process vessels is the basis of the energy and material balances that may be done to calculate the vessel properties such as temperature, composition, and pressure. Often the dynamics associated with stream processing i.e. actuators and processing elements included in a pipe, duct or conveyor **are much faster than those associated with processing vessels**. For the simulation to be done is DeltaV, a sequential modular approach will be implemented based on taking advantage of this difference in dynamics. Specifically, for the purpose of dynamic simulation, the conditions associated with processing vessels may **be assumed constant** for during the multiple iterations of stream processing simulation in which the flow and pressure is established between vessels, as illustrated below. Similarly, the processing associated with process vessels may be done assuming that the inlet and outlet flows to the vessel are constant for each iteration of the simulation.



Such an approach allows the simulation of streams and processing units to be divided into smaller units that are decoupled through this "tearing" in connections.

Each element in the stream will calculate its outlet pressure based on its inlet flow and pressure. The calculated outlet pressure by the final element for an assumed flow in a stream will be used in conjunction with the downstream and upstream vessel pressure to calculate a new guess for the flow to force convergence of the stream outlet to match the pressure of the downstream vessel. This new flow guess is then used in the next iteration of the stream calculation. A convergence block will be introduced after the last element of the stream to calculate the flow guess to force convergence as illustrated in the following example.



The elements that are contained after a pressure source within a stream, such as valve, heat exchanges, and pipe, are characterized by the fact that the flow through any one of these elements is related to the differential pressure across the element in the following manner:

$$\text{Element 1 Differential pressure} = K_1 * (\text{Flow})^2 \quad (1)$$

Where K_1 is the characteristic resistance for that element

For elements such as a regulating valve, the characteristic resistance is a function of stem position. However, during the convergence iterations for a stream, the resistance associated with each element may be considered to be constant. Thus, assuming the flow through each element of the stream is the same, then the total pressure drop introduced by the N elements in the stream is:

$$\begin{aligned} \text{Total stream pressure drop} &= (K_1 + K_2 + \dots + K_N) * (\text{Flow})^2 \\ &= K_{\text{Total}} * (\text{Flow})^2 \end{aligned} \quad (2)$$

Based on this fact, then it is possible to calculate the system resistance for a known pressure drop and flow.

$$K_{\text{Total}} = \text{Total stream pressure drop} / (\text{Flow})^2 \quad (3)$$

During the first iteration of the stream calculation, the stream element will be evaluated in the same order as the flow through the stream. Based on its inlet pressure, each element will calculate its outlet pressure for the assumed stream flow. On the first iteration, the last stream flow will be used. Based on the upstream pressure and the calculated outlet pressure of the last element in the stream, the total system resistance will be calculated. Using this calculated system resistance and the pressure differential between the stream pressure source (vessel or pump) and the downstream vessel, the a new guess for flow may be calculated and then used in the next iteration.

$$\text{New Flow Guess} = \text{SqRt} ((\text{Source Press} - \text{downstream vessel press}) / K_{\text{Total}}) \quad (4)$$

Using this procedure, the stream flow-pressure calculation will normally **converge within two iterations (valve, heat exchanges, pipe) or three iterations (multiple elements with pump as pressure source).**

In this section, the structure of steam elements will be examined based on this approach in resolving pressure-flow interaction and their interaction with process vessels.

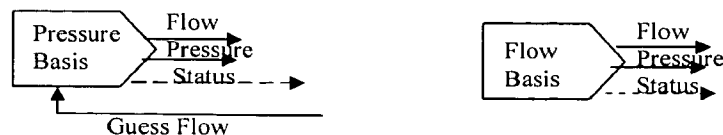
3.1 Stream Element

As part of the stream element configuration, the user may define the basis stream simulation to be

Pressure based – a constant pressure is provided. The flow through the stream will be provided by the converge block.

Flow Based – a constant flow and pressure output is provided independent of the converge block or the downstream conditions.

In most applications, the stream element will be configured for pressure based i.e. will be the configured default value will be pressure based. The communication of pressure and flow properties associated with the stream element may be represented as illustrated below.



Stream Element May Be a Pressure or Flow Source

The connection status passed between Stream Element and a processing element will reflect the basis associated with a connection i.e. pressure or flow.

<u>Out Status</u>	<u>Description</u>
Good_F	Flow Basis – pass the flow through the steam
Good_P_F	Pressure Basis – calculate downstream pressure based on flow, closed valve elements may set flow to zero.

Based on this status, the downstream element will automatically calculate pressure or flow. If a stream element is configured as a reference to a stream in another process module, then this stream will automatically propagate the connection status of the referenced stream.

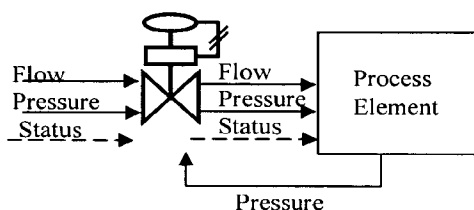
3.2 Regulating Valve

A regulating valve element that is open will calculate its outlet flow and pressure based on its inputs status

<u>Input Status</u>	<u>Calculation done</u>
Good_F	The inlet flow and pressure are propagated by the valve independent of the stem position.
Good_P_F	The outlet pressure is calculated based on the inlet pressure and flow

If a regulating valve stem position is below its minimal controllable flow and the inputs status is **Good_P_F**, then the outlet flow will be set to zero, outlet pressure will reflect the downstream pressure, and the outlet status set to a value that indicates no flow through the steam is possible:

<u>Outlet Status</u>	<u>Condition</u>
Good_P	Pressure basis stream (input status is Good_P_F) but valve is closed.



The valve calculation will assume a static mass and energy balance i.e. the flow, chemical composition and temperature in and out of the valve are the same. The calculations done will be based on the flow through the valve being related to the valve stem position, and differential pressure across the valve in the following manner.

$$F = C_{VMax} * f(l) * \text{Sqr}(dP_V / G_S) \quad (5)$$

Where:

F = Flow in GPM e.g 250

$f(I)$ = Flow characteristics

Linear: $f(I) = I$

Quick Opening: $f(I) = \text{SquareRoot}(I)$

Equal Percentage: $f(I) = R^{(I-1)}$

I = % Valve Open/100

$C_{V\text{Max}}$ = Valve coefficient at 100% open, GPM/psi

G_S = The specific gravity of the fluid (1 at 60 degF)

dP_V = Pressure Differential in psi

R = Valve rangeability, Ratio of the maximum to minimum controllable flow through the valve

Thus, if the valve is open and the input indicated a Good_P_F status, then downstream pressure will be calculated as follows based on the inlet flow

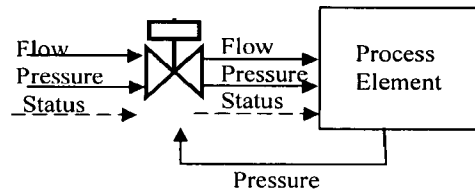
$$\text{Outlet Pressure} = \text{Input pressure} - G_S (F / (C_{V\text{Max}} * f(I)))^2 \quad (6)$$

Note: The Calculation above does not include viscosity correction factor, assumes Reynolds number is greater than 3500. Need to do calculation in SI units.

3.3 Blocking Valve

The inlet status, pressure and flow will be propagated by an open blocking valve. When the blocking valve is closed, then the action taken depends on the inlet status. If the status indicates a flow basis stream i.e. status is Good_F, then the flow, pressure and status will be propagated even if the valve is close. If the inlet status is Good_P_F or Good_P and the valve is closed, then the out flow will be set to zero, the outlet pressure reflects the downstream pressure, and the out status set to Good_P.

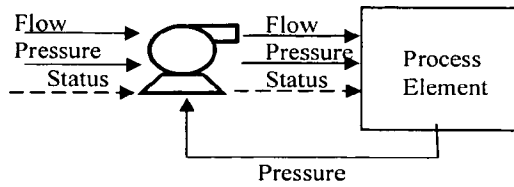
The user may specify the rate at which the valve closes. During this transition, the changing valve position will be reflected in the pressure for a pressure basis stream..



The calculations associated with the blocking valve will assume that there is no pressure drop across an open blocking valve.

3.4 Fixed Speed Centrifugal Pump

A pump may be the pressure source for a stream i.e. the start of a **pressure basis stream**. The discharge pressure will be determined based on the suction pressure and the discharge flow. When the pump is shut off, then the discharge pressure will revert to the pressure of the downstream element.



The fixed speed pump calculation will assume a static mass and energy balance i.e. the flow, chemical composition and temperature in and out of the valve are the same. The calculations done will be based on the flow and differential pressure across the pump being related in the following manner.

$$F = F_{Max} - (dP/K_{Pump})^2 \quad (7)$$

Where F = Pump discharge flow

F_{Max} = Maximum block possible (zero system pressure loss)

dP = Discharge pressure – Inlet (suction) pressure

K_{Pump} = Pump constant

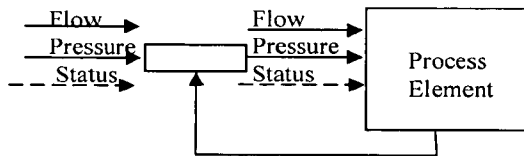
Thus, when the pump motor is ON, then the discharge pressure may be calculated as follows:

$$\text{Outlet Pressure} = \text{Input Pressure} + K_{Pump} * \text{SQRT}(F_{Max} - F) \quad (8)$$

If the pump is off, then an outlet flow of zero and status of Good_P will be calculated.

3.5 Pipe Segment

The pipe connection between processing elements will normally be assumed to have no impact i.e. status and properties are propagated. However, if the user configures the a length and diameter and the inlet status is Good_P_F, then the pressure loss introduced the flow through the pipe will be calculated based on the inlet flow and status.



The flow and pressure drop across the pipe may be calculated as follows:

$$Dp = K_2 * F^2$$

Where:

F = Pipe flow

D_p = Differential pressure drop across a pipe

K_2 = Pipe constant (characteristic of pipe diameter and length)

Thus, the outlet pressure may be calculated as follows:

$$\text{Outlet Pressure} = \text{Inlet pressure} - K_2 * F^2$$

When a pipe is configured, then the composition properties associated with the output connection will be the same as the input (assumed plug flow) but dynamically delayed based on the flow rate and size and length of pipe.

$$\text{Composition Out}(t) = \text{Composition In}(t - t_{\text{Delay}})$$

Where $t_{\text{Delay}} = \text{Pipe Volume} / \text{Flow Rate}$

3.6 Heat Exchanger

Since the outlet flow of a heat exchanger is the same as the outlet flow and can change quickly based on upstream and downstream condition, then the heat exchanger will be treated as a processing element in the connections between vessels.

The pressure flow relationship in the process steam in the process flow to and from the heater can be described as follows.

$$Dp = K_2 * F^2$$

Where:

F= Process flow

Dp = Differential pressure drop across the heat exchanger

K₂ = Heat exchanger constant (characteristic of exchanger size)

The outlet pressure may be calculated as follows:

$$\text{Outlet Pressure} = \text{Inlet pressure} - K_2 * F^2$$

The outlet temperature may be calculated as follows assuming liquid flow through a single pass, single shell pass counter current heat exchanger.

$$N1 = U A / W1 Cp1$$

$$N2 = U A / W2 Cp2$$

$$T2out = [[Exp(N2-N1)-1] T1in + [1-N1/N2] T2in] / [Exp(N2-N1) - N1/N2]$$

$$T1out = [Exp(N2-N1) [1-N1/N2] T1in + N1/N2 [Exp(N2-N1)-1] T2in] / [Exp(N2-N1)-N1/N2]$$

Where

U = Heat transfer Coefficient - W/(m² K)

A = Heat Exchanger Area – m²

W1 = Mass Flow Tube Side - kg/s

W2 = Mass Flow Shell Side – kg/s

Cp1 = Specific heat of liquid in Tube – kj/(kg K)

Cp2 = Specific heat of liquid in Shell – kj/(kg K)

T1in and T1out = Tube inlet and outlet temperature – C

T2in and T2out = Shell inlet and outlet temperature – C

3.7 Vessel Simulation

The simulation of each process vessel will be done assuming that the inlet and outlet flow are constant for the vessel calculation. In this section, we consider some basic vessels to illustrate this concept.

3.7.1 Atmospheric Tank

The simulation of an atmospheric tank will include the calculation of the tank level. This level will be calculated at each step of the simulation assuming that the inlet and outlet flows are constant. Checks will be made to insure the calculated level remains within the tank height and does not go below the height of the pump out connection. Also, based on the specific gravity of the material in the tank, a liquid head pressure will be provided for each input and output connection. For the output connections, the associated status will indicate if the liquid level is above the connection point.

$$\text{Level}_{\text{New}} = \text{Level}_{\text{Old}} + (\text{sum of input flows} - \text{sum of outlet flows}) * (\text{execution period} / 60) / K$$

If Level > Tank Height Then

Level = Tanks Height

Endif

If Level < Height of Pumpout Connection

Level = Height of Pumpout Connection.

Outstatus = Good_P

Endif

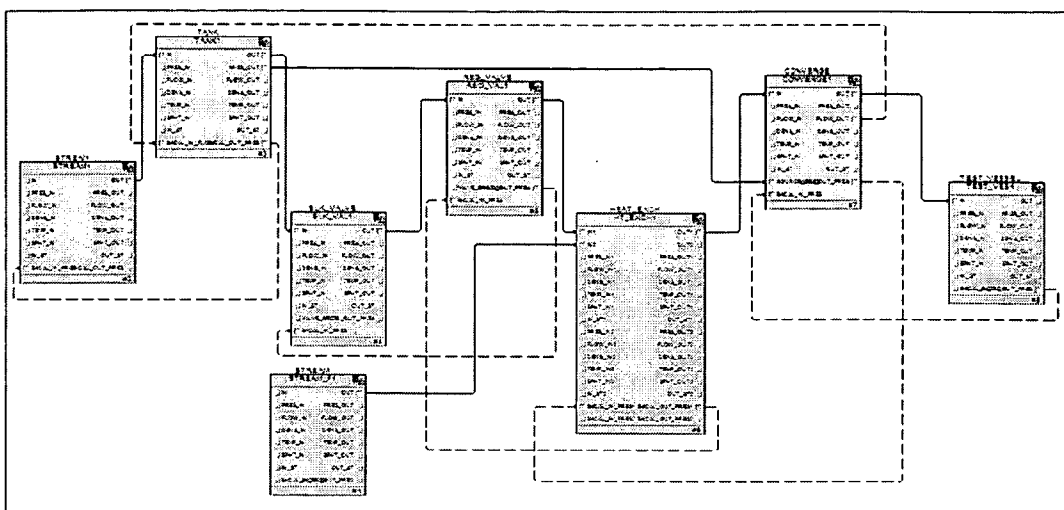
Where K = Tank constant, Cubic Liters/Liter

4 Example Implementation

Composite blocks may be created in DeltaV to test and demonstrate the elements defined process simulation of elements in a stream, as discussed in section 3. In this section, we show an example implementation of these simulation elements. These examples address the following elements:

- Stream
- Constant Speed Centrifugal pump
- Regulation Valve
- Blocking Valve
- Pipe
- Converge Block
- Test Vessel
- Tank
- Heat Exchanger
- Variable Speed Pump

The test vessel and the stream element to create the upstream pressure source. An example of how these composites may be used to simulation a stream consisting of a stream element, regulation valve and vessel is shown below:



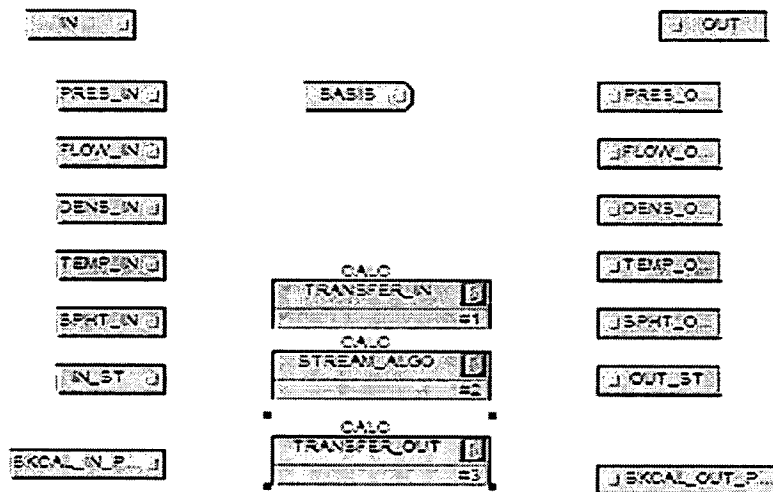
4.1 Named Set Definitions

To make it easier to use enumerated parameter in the process simulation block definition, the following named sets were created in this example.

Name Set	Enumerations	Value	Description
BASIS	Pressure	0	The calculation basis
	Flow	1	
PROC_ST	No_Connection	0	Connection Status Definitions
	Good_P_F	1	Pressure and Flow may vary
	Good_P	2	Pressure may vary, Flow = 0
	Good_F	3	Flow is constant
VALVE_CHAR	Linear	0	Regulation Valve characteristics
	Equal Percentage	1	
	Quick Opening	2	
BLK_VALVE	Closed	0	States of a blocking valve
	Open	1	
Motor	Off	0	Motor state
	On	1	

4.2 Named Basic Properties – Use of Array Parameter

To minimize the number of connections between blocks, an IN and OUT parameter is provided in each block. These parameters are defined as floating arrays (6x1) so that the five basic properties of each connection plus the status may be exchanged with a single connection. The properties associated with a block output are put into OUT array parameter using a transfer output Calc/Logic block. Similarly, the values in the IN array parameter are unpacked and put into the block input parameters by the Transfer Input Calc/Logic block. A typical example of how this is done in the construction of a composite block is shown in the following:



Example of IN and OUT for Transfer of connection Properties

The expression for the Transfer Input Calculation block is shown below:

```

1 IF (^/IN'[6][1] != 0) THEN      (* If not connected, then this is a source*)
2 ^/PRES_IN' := ^/IN'[1][1];
3 ^/FLOW_IN' := ^/IN'[2][1];
4 ^/DENS_IN' := ^/IN'[3][1];
5 ^/TEMP_IN' := ^/IN'[4][1];
6 ^/SPHT_IN' := ^/IN'[5][1];
7 ^/IN_ST' := ^/IN'[6][1];
8 ENDIF;

```

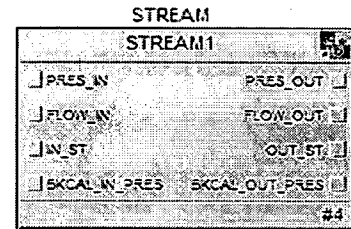
The expression for the Transfer Output Calculation block is shown below:

```
1 ^/OUT'[1][1] := ^/PRES_OUT';  
2 ^/OUT'[2][1] := ^/FLOW_OUT';  
3 ^/OUT'[3][1] := ^/DENS_OUT';  
4 ^/OUT'[4][1] := ^/TEMP_OUT';  
5 ^/OUT'[5][1] := ^/SPHT_OUT';  
6 ^/OUT'[6][1] := ^/OUT_ST';
```

The input and output parameters are currently also shown as connection points just to allow the values to be easily viewed for checkout – should not be connected to in building example.. In the future most of these input and output parameters will be change to internal inputs and output that will be viewed from the left hand pane when the block is selected.

4.3 Named Steam Element

The stream element is defined as a composite template. This block utilized the following parameters and algorithm.



4.3.1 Stream Parameters

Parameter	Default	Connection type	Parameter type
BASIS	Pressure	Internal read only	Named Set
BKCAL_IN_P...	0	Input	Floating point
BKCAL_OUT...	0	Output	Floating point
DENS_IN	1000	Input	Floating point
DENS_OUT	0	Output	Floating point
FLOW_IN	0	Input	Floating point
FLOW_OUT	0	Output	Floating point
IN		Input	Floating point array
IN_ST	No_Connect...	Input	Named Set
OUT		Output	Floating point array
OUT_ST	No_Connect...	Output	Named Set
PRES_IN	0	Input	Floating point
PRES_OUT	0	Output	Floating point
SPHT_IN	4.184	Input	Floating point
SPHT_OUT	0	Output	Floating point
TEMP_IN	30	Input	Floating point
TEMP_OUT	0	Output	Floating point

4.3.2 Stream Algorithm

```

1) ^/PRES_OUT' := ^/PRES_IN';
2) ^/FLOW_OUT' := ^/FLOW_IN';
3) ^/DENS_OUT' := ^/DENS_IN';
4) ^/TEMP_OUT' := ^/TEMP_IN';
5) ^/SPHT_OUT' := ^/SPHT_IN';
6) IF ^/IN_ST' = 'PROC_ST:No_Connection' THEN
7)   IF ^/BASIS' = 'BASIS:Pressure' THEN
8)     ^/OUT_ST' := 'PROC_ST:Good_P_F';
9)   ELSE
10)    ^/OUT_ST' := 'PROC_ST:Good_F';
11)   ENDIF;
12) ELSE
13)   ^/OUT_ST' := ^/IN_ST';
14) ENDIF;
15) ^/BKCAL_OUT_PRES' := ^/BKCAL_IN_PRES';

```

(*Stream always propagates properties*)

(* If not connected, then this is a source*)

(*Source of Press and Propagate Flow*)

(*Source of Flow and Propagate Pressure*)

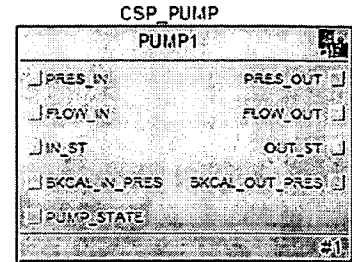
(*Acting as a connection, Propagate IN_ST*)

4.4 Constant Speed Centrifugal Pump Element

The pump element is defined as a composite template. This block utilized the following parameters and algorithm.

4.4.1 Pump Parameters

Parameter	Default	Connection type	Parameter type
BKCAL_IN_P...	0	Input	Floating point
BKCAL_OUT...	0	Output	Floating point
DENS_IN	1000	Input	Floating point
DENS_OUT	0	Output	Floating point
FLOW_IN	0	Input	Floating point
FLOW_MAX	31.48	Internal read only	Floating point
FLOW_OUT	0	Output	Floating point
IN		Input	Floating point array
IN_ST	No Connect...	Input	Named Set
OUT		Output	Floating point array
OUT_ST	No Connect...	Output	Named Set
PRES_IN	0	Input	Floating point
PRES_OUT	0	Output	Floating point
PUMP_CONST	80	Internal read only	Floating point
PUMP_STATE	On	Input	Named Set
SPHT_IN	4.184	Input	Floating point
SPHT_OUT	0	Output	Floating point
TEMP_IN	30	Input	Floating point
TEMP_OUT	0	Output	Floating point



4.4.2 Pump Algorithm

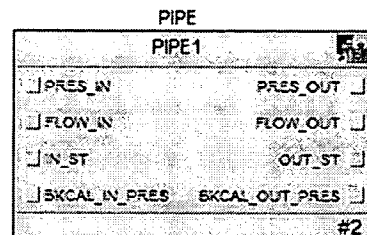
```

1 IF (IN_ST = 'PROC_ST:No_Connection') THEN (* If not connected, then propagate value and status *)
2   ~OUT_ST := 'PROC_ST:Good_P_F';
3   ~PRES_OUT := ~PRES_IN;
4   ~FLOW_OUT := ~FLOW_IN;
5   ~BKCAL_OUT_PRES := ~BKCAL_IN_PRES;
6 ELSE
7   IF (IN_ST = 'PROC_ST:Good_F') THEN (* Constant flow, propagate pressure flow status *)
8     ~OUT_ST := IN_ST;
9     ~PRES_OUT := ~PRES_IN;
10    ~FLOW_OUT := ~FLOW_IN;
11    ~BKCAL_OUT_PRES := ~BKCAL_IN_PRES;
12  ELSE
13    IF (~PUMP_STATE = 'MOTOR:Off') THEN (* Determine if pump running *)
14      ~FLOW_OUT := 0.0; (* Motor off no flow possible *)
15      ~PRES_OUT := ~BKCAL_IN_PRES;
16      ~BKCAL_OUT_PRES := ~PRES_IN;
17      ~OUT_ST := 'PROC_ST:Good_P_F';
18    ELSE
19      IF (IN_ST = 'PROC_ST:Good_P') THEN (* No flow upstream, propagate press *)
20        ~FLOW_OUT := 0.0;
21        ~PRES_OUT := ~PRES_IN;
22        ~BKCAL_OUT_PRES := ~PRES_IN;
23        ~OUT_ST := IN_ST;
24      ELSE
25        IF (~FLOW_IN > ~FLOW_MAX) THEN
26          ~FLOW_OUT := ~FLOW_MAX;
27        ELSE
28          ~FLOW_OUT := ~FLOW_IN;
29        ENDIF;
30        FIFactor := (~FLOW_MAX * ~FLOW_OUT); (* Calculate discharge pressure *)
31        IF (FIFactor < 0.0) THEN
32          FIFactor := 0.0;
33        ENDIF;
34        ~PRES_OUT := ~PRES_IN + (~PUMP_CONST) * SQRT(FIFactor);
35        ~BKCAL_OUT_PRES := ~PRES_IN;
36        ~OUT_ST := IN_ST;
37      ENDIF;
38    ENDIF;
39  ENDIF;
40 ENDIF;
41 ~DENS_OUT := ~DENS_IN;
42 ~TEMP_OUT := ~TEMP_IN;
43 ~SPHT_OUT := ~SPHT_IN;

```

4.5 Pipe Element

The pipe element is defined as a composite template. This block utilized the following parameters and algorithm.



4.5.1 Pipe Parameters

Parameter	Default	Connection type	Parameter type
BKCAL_IN_P...	0	Input	Floating point
BKCAL_OUT...	0	Output	Floating point
DENS_IN	1000	Input	Floating point
DENS_OUT	0	Output	Floating point
FLOW_IN	0	Input	Floating point
FLOW_OUT	0	Output	Floating point
IN		Input	Floating point array
IN_ST	No_Connect...	Input	Named Set
OUT		Output	Floating point array
OUT_ST	No_Connect...	Output	Named Set
PIPE_CONST	0	Internal read only	Floating point
PRES_IN	0	Input	Floating point
PRES_OUT	0	Output	Floating point
SPHT_IN	4.184	Input	Floating point
SPHT_OUT	0	Output	Floating point
TEMP_IN	30	Input	Floating point
TEMP_OUT	0	Output	Floating point

4.5.2 Pipe Algorithm

```

1 IF (^/IN_ST' = 'PROC_ST:No_Connection') THEN (* If not connected, then propagate value and status*)
2   ^/OUT_ST' := 'PROC_ST:Good_P_F';
3   ^/PRES_OUT' := ^/PRES_IN';
4   ^/FLOW_OUT' := ^/FLOW_IN';
5   ^/BKCAL_OUT_PRES' := ^/BKCAL_IN_PRES';
6 ELSE
7   IF (^/IN_ST' = 'PROC_ST:Good_F') THEN (* Constant flow, propagate pressure flow status *)
8     ^/OUT_ST' := ^/IN_ST';
9     ^/PRES_OUT' := ^/PRES_IN';
10    ^/FLOW_OUT' := ^/FLOW_IN';
11    ^/BKCAL_OUT_PRES' := ^/BKCAL_IN_PRES';
12  ELSE
13    IF (^/IN_ST' = 'PROC_ST:Good_P') THEN (* No flow upstream, propagate press *)
14      ^/FLOW_OUT' := 0.0;
15      ^/PRES_OUT' := ^/PRES_IN';
16      ^/BKCAL_OUT_PRES' := ^/BKCAL_IN_PRES';
17      ^/OUT_ST' := ^/IN_ST';
18    ELSE
19      ^/FLOW_OUT' := ^/FLOW_IN'; (* Calculate pipe outlet pressure *)
20      ^/PRES_OUT' := ^/PRES_IN' - ((^/FLOW_IN' * ^/FLOW_IN') * ^/PIPE_CONST');
21      ^/BKCAL_OUT_PRES' := ^/PRES_IN';
22      ^/OUT_ST' := ^/IN_ST';
23    ENDIF;
24  ENDIF;
25 ENDIF;
26 ^/DENS_OUT' := ^/DENS_IN';
27 ^/TEMP_OUT' := ^/TEMP_IN';
28 ^/SPHT_OUT' := ^/SPHT_IN';

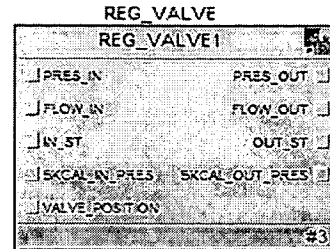
```

4.6 Regulating Valve Element

The regulating valve element is defined as a composite template. This block utilized the following parameters and algorithm.

4.6.1 Regulating Valve Parameters

Parameter	Default	Connection type	Parameter type
BKCAL_IN_P...	0	Input	Floating point
BKCAL_OUT...	0	Output	Floating point
CHARACTERI...	Linear	Internal read only	Named Set
CV	3.3	Internal read only	Floating point
DENS_IN	1000	Input	Floating point
DENS_OUT	0	Output	Floating point
FLOW_IN	0	Input	Floating point
FLOW_OUT	0	Output	Floating point
IN		Input	Floating point array
IN_ST	No_Connect...	Input	Named Set
OUT		Output	Floating point array
OUT_ST	No_Connect...	Output	Named Set
PRES_IN	0	Input	Floating point
PRES_OUT	0	Output	Floating point
RANGEABILITY	30	Internal read only	Floating point
SPECIFIC_GR...	1	Internal read only	Floating point
SPHT_IN	4.184	Input	Floating point
SPHT_OUT	0	Output	Floating point
TEMP_IN	30	Input	Floating point
TEMP_OUT	0	Output	Floating point
VALVE_POSI...	0	Input	Floating point



4.6.2 Regulating Valve Algorithm

```

1) ^/SPECIFIC_GRAVITY := ^/DENS_IN / 997.948; (* Divide by kg water/m3 at 60 F *)
2) IF (^/IN_ST = 'PROC_ST:No_Connection') THEN (* If not connected, then propagate value and stat
3) ^/OUT_ST := 'PROC_ST:Good_P_F';
4) ^/PRES_OUT := ^/PRES_IN;
5) ^/FLOW_OUT := ^/FLOW_IN;
6) ^/BKCAL_OUT_PRES := ^/BKCAL_IN_PRES;
7) ELSE
8) IF (^/IN_ST = 'PROC_ST:Good_F') THEN (* Constant flow, propagate pressure flow status *)
9) ^/OUT_ST := ^/IN_ST;
10) ^/PRES_OUT := ^/PRES_IN;
11) ^/FLOW_OUT := ^/FLOW_IN;
12) ^/BKCAL_OUT_PRES := ^/BKCAL_IN_PRES;
13) ELSE
14) MinimumOpen := (100.0 / ^/RANGEABILITY); (* Minimum opening for flow *)
15) IF (^/VALVE_POSITION < MinimumOpen) THEN (* Determine if valve is closed *)
16) ^/FLOW_OUT := 0.0; (* Valve closed, no flow possible *)
17) ^/PRES_OUT := ^/BKCAL_IN_PRES;
18) ^/BKCAL_OUT_PRES := ^/PRES_IN;
19) ^/OUT_ST := 'PROC_ST:Good_P';
20) ELSE
21) IF (^/IN_ST = 'PROC_ST:Good_P') THEN
22) ^/FLOW_OUT := 0.0; (* No flow upstream, propagate press *)
23) ^/PRES_OUT := ^/PRES_IN;
24) ^/BKCAL_OUT_PRES := ^/PRES_IN;
25) ^/OUT_ST := ^/IN_ST;
26) ELSE
27) ^/FLOW_OUT := ^/FLOW_IN;
28) FractionOpen := (^/VALVE_POSITION / 100.0); (* Determine downstm press *)
29) IF (^/CHARACTERISTICS = 'VALVE_CHAR:Linear') THEN
30) FlowChar := FractionOpen; (* Linear *)
31) ELSE
32) IF (^/CHARACTERISTICS = 'VALVE_CHAR:Quick Opening') THEN
33) FlowChar := SQRT(FractionOpen); (* Quick Opening *)
34) ELSE
35) Power := (FractionOpen - 1.0); (* Equal Percentage *)
36) FlowChar := EXP(1 - ^/RANGEABILITY * Power);
37) ENDIF;
38) ENDIF;
39) FIFactor := (^/FLOW_IN / (^/CV * FlowChar));
40) ^/PRES_OUT := (^/PRES_IN * FIFactor * FIFactor * ^/SPECIFIC_GRAVITY);
41) ^/BKCAL_OUT_PRES := ^/PRES_IN;
42) ^/OUT_ST := ^/IN_ST;
43) ENDIF;
44) ENDIF;
45) ENDIF;
46) ^/DENS_OUT := ^/DENS_IN;
47) ^/TEMP_OUT := ^/TEMP_IN;
48) ^/SPHT_OUT := ^/SPHT_IN;

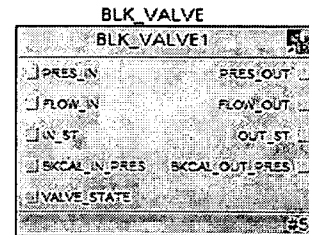
```

4.7 Blocking Valve Element

The blocking valve element is defined as a composite template. This block utilized the following parameters and algorithm.

4.7.1 Blocking Valve Parameters

Parameter	Default	Connection type	Parameter type
BKCAL_IN_P...	0	Input	Floating point
BKCAL_OUT...	0	Output	Floating point
DENS_IN	1000	Input	Floating point
DENS_OUT	0	Output	Floating point
FLOW_IN	0	Input	Floating point
FLOW_OUT	0	Output	Floating point
IN		Input	Floating point array
IN_ST	No_Connect...	Input	Named Set
OUT		Output	Floating point array
OUT_ST	No_Connect...	Output	Named Set
PRES_IN	0	Input	Floating point
PRES_OUT	0	Output	Floating point
SPHT_IN	4.184	Input	Floating point
SPHT_OUT	0	Output	Floating point
TEMP_IN	30	Input	Floating point
TEMP_OUT	0	Output	Floating point
VALVE_STATE	Open	Input	Named Set



4.7.2 Blocking Valve Algorithm

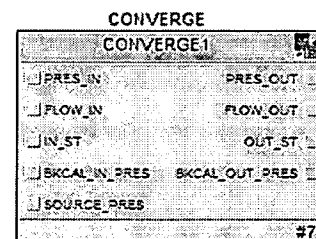
```

1 IF (~IN_ST = PROC_ST:No_Connection') THEN (* If not connected, then propagate value and status *)
2   ~OUT_ST := PROC_ST:Good_P_F;
3   ~PRES_OUT := ~PRES_IN;
4   ~FLOW_OUT := ~FLOW_IN;
5   ~BKCAL_OUT_PRES := ~BKCAL_IN_PRES;
6 ELSE
7   IF (~IN_ST = PROC_ST:Good_F) THEN (* Constant flow, propagate pressure flow status *)
8     ~OUT_ST := ~IN_ST;
9     ~PRES_OUT := ~PRES_IN;
10    ~FLOW_OUT := ~FLOW_IN;
11    ~BKCAL_OUT_PRES := ~BKCAL_IN_PRES;
12   ELSE
13     IF (~VALVE_STATE = 'BLOCK_VALVE:Close') THEN (* Determine if valve is closed *)
14       ~FLOW_OUT := 0.0; (* Valve closed, no flow possible *)
15       ~PRES_OUT := ~BKCAL_IN_PRES;
16       ~BKCAL_OUT_PRES := ~PRES_IN;
17       ~OUT_ST := PROC_ST:Good_P;
18     ELSE
19       IF (~IN_ST = PROC_ST:Good_P) THEN (* No flow upstream, propagate press *)
20         ~FLOW_OUT := 0.0;
21         ~PRES_OUT := ~PRES_IN;
22         ~BKCAL_OUT_PRES := ~PRES_IN;
23         ~OUT_ST := ~IN_ST;
24       ELSE
25         ~FLOW_OUT := ~FLOW_IN; (* Valve open, propagate flow and press *)
26         ~PRES_OUT := ~PRES_IN;
27         ~BKCAL_OUT_PRES := ~PRES_IN;
28         ~OUT_ST := ~IN_ST;
29       ENDIF;
30     ENDIF;
31   ENDIF;
32 ENDIF;
33 ~DENS_OUT := ~DENS_IN;
34 ~TEMP_OUT := ~TEMP_IN;
35 ~SPHT_OUT := ~SPHT_IN;

```

4.8 Convergence Element

The convergence element is defined as a composite template. This block utilized the following parameters and algorithm.



4.8.1 Convergence Parameters

Parameter	Default	Connection type	Parameter type
BKCAL_IN_P...	0	Input	Floating point
BKCAL_OUT...	0	Output	Floating point
DENS_IN	1000	Input	Floating point
DENS_OUT	0	Output	Floating point
FLOW_IN	0	Input	Floating point
FLOW_OUT	0	Output	Floating point
IN		Input	Floating point array
IN_ST	No_Connect...	Input	Named Set
OUT		Output	Floating point array
OUT_ST	No_Connect...	Output	Named Set
PRES_IN	0	Input	Floating point
PRES_OUT	0	Output	Floating point
SOURCE_PRES	0	Input	Floating point
SPHT_IN	4.184	Input	Floating point
SPHT_OUT	0	Output	Floating point
TEMP_IN	30	Input	Floating point
TEMP_OUT	0	Output	Floating point

4.8.2 Convergence Algorithm

```

1 IF (^/IN_ST' = 'PROC_ST:No_Connection') THEN ("If not connected, then propagate value and status")
2   ^/OUT_ST' := 'PROC_ST:Good_P_F';
3   ^/PRES_OUT' := ^/PRES_IN';
4   ^/FLOW_OUT' := ^/FLOW_IN';
5   ^/BKCAL_OUT_PRES' := ^/BKCAL_IN_PRES';
6 ELSE
7   IF (^/IN_ST' = 'PROC_ST:Good_F') THEN ("Constant flow, propagate pressure flow status")
8     ^/OUT_ST' := ^/IN_ST';
9     ^/PRES_OUT' := ^/PRES_IN';
10    ^/FLOW_OUT' := ^/FLOW_IN';
11    ^/BKCAL_OUT_PRES' := ^/BKCAL_IN_PRES';
12  ELSE
13    IF (^/IN_ST' = 'PROC_ST:Good_P') THEN
14      ^/FLOW_OUT' := 0.0; ("No flow upstream, propagate press")
15      ^/PRES_OUT' := ^/PRES_IN';
16      ^/BKCAL_OUT_PRES' := ^/BKCAL_IN_PRES';
17      ^/OUT_ST' := ^/IN_ST';
18    ELSE
19      IF (^/FLOW_IN' <= 0.0) THEN
20        ^/FLOW_OUT' := 1; ("Flow must be GT 0 to calc resistance, try again using 1")
21      ELSE
22        CalcPresDrop := ^/SOURCE_PRES' - ^/PRES_IN'; ("DP for assumed flow")
23        Resistance := CalcPresDrop / (^/FLOW_IN' * ^/FLOW_IN');
24        FIFactor := (^/SOURCE_PRES' - ^/BKCAL_IN_PRES') / Resistance;
25        ^/FLOW_OUT' := SQRT ( FIFactor ); ("New guess for flow")
26      ENDIF;
27      ^/PRES_OUT' := ^/BKCAL_IN_PRES';
28      ^/BKCAL_OUT_PRES' := ^/BKCAL_IN_PRES';
29      ^/OUT_ST' := ^/IN_ST';
30    ENDIF;
31  ENDIF;
32 ENDIF;
33 ^/DENS_OUT' := ^/DENS_IN';
34 ^/TEMP_OUT' := ^/TEMP_IN';
35 ^/SPHT_OUT' := ^/SPHT_IN';

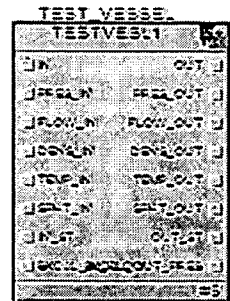
```

4.9 Test Vessel Element

The test vessel element is defined as a composite template. This block utilized the following parameters and algorithm.

4.9.1 Test Vessel Parameters

Parameter	Default	Connection type	Parameter type
BASIS	Pressure	Internal read only	Named Set
BKCAL_IN_F...	0	Input	Floating point
BKCAL_OUT...	0	Output	Floating point
DENS_IN	1000	Input	Floating point
DENS_OUT	0	Output	Floating point
FLOW_IN	0	Input	Floating point
FLOW_OUT	0	Output	Floating point
IN		Input	Floating point array
IN_ST	No_Connect...	Input	Named Set
OUT		Output	Floating point array
OUT_ST	Good_P_F	Output	Named Set
PRES_IN	0	Input	Floating point
PRES_OUT	0	Output	Floating point
SPHT_IN	4.184	Input	Floating point
SPHT_OUT	0	Output	Floating point
TANK_PRES	790	Internal read only	Floating point
TEMP_IN	30	Input	Floating point
TEMP_OUT	0	Output	Floating point



4.9.2 Test Vessel Algorithm

```

1 ^/PRES_OUT' := ^/TANK_PRES';
2 ^/BKCAL_OUT_PRES' := ^/TANK_PRES';
3 IF ( ^/BASIS' = 'BASIS:Pressure' ) THEN
4   ^/FLOW_OUT' := ^/BKCAL_IN_FLOW'; (* Use new calculated flow *)
5   ^/OUT_ST' := 'PROC_ST:Good_P_F';
6 ELSE
7   ^/OUT_ST' := 'PROC_ST:Good_F'; (* Flow basis, use entered flow value *)
8 ENDIF;

```

4.10 Tank Element

The tank element is defined as a composite template. This block utilized the following parameters and algorithm.

4.10.1 Tank Parameters

Parameter	Default	Connection type	Parameter type
BASIS	Pressure	Internal read only	Named Set
BKCAL_IN_F...	0	Input	Floating point
BKCAL_OUT...	0	Output	Floating point
DENS_IN	0	Input	Floating point
DENS_OUT	0	Output	Floating point
FLOW_IN	0	Input	Floating point
FLOW_OUT	0	Output	Floating point
IN		Input	Floating point array
IN_ST	No_Connect...	Input	Named Set
LIQ_DENS	0	Internal read only	Floating point
LIQ_LEVEL	0	Internal read only	Floating point
LIQ_MASS	0	Internal read only	Floating point
LIQ_SPHT	0	Internal read only	Floating point
LIQ_TEMP	0	Internal read only	Floating point
MAX_LEVEL	10	Internal read only	Floating point
OUT		Output	Floating point array
OUT_ST	Good_P_F	Output	Named Set
PERIOD_EXEC	1	Internal read only	Floating point
POS_IN	1	Internal read only	Floating point
POS_OUT	2	Internal read only	Floating point
PRES_IN	0	Input	Floating point
PRES_OUT	0	Output	Floating point
SPHT_IN	0	Input	Floating point
SPHT_OUT	0	Output	Floating point
TEMP_IN	0	Input	Floating point
TEMP_OUT	0	Output	Floating point
TNK_CONS	10	Internal read only	Floating point

4.10.2 Tank Algorithm

```

1 IF ( ^/IN_ST' = 'PROC_ST:No_Connection' ) THEN (* If not connected, then use o value *)
2   ^/FLOW_IN' := 0.0;
3 ELSE
4   IF ( ^/IN_ST' = 'PROC_ST:Good_F' ) THEN (* Constant flow, propagate pressure flow status *)
5     ^/OUT_ST' := ^/IN_ST';
6     ^/FLOW_OUT' := ^/FLOW_IN';
7   ELSE
8     IF ( ^/LIQ_LEVEL' < ^/POS_OUT' ) THEN (* Determine if level is below outlet *)
9       ^/OUT_ST' := 'PROC_ST:Good_P';
10      ^/FLOW_OUT' := 0.0; (* No flow possible *)
11    ELSE
12      ^/OUT_ST' := 'PROC_ST:Good_P_F';
13      ^/FLOW_OUT' := ^/BKCAL_IN_FLOW';
14    ENDIF;
15  ENDIF;
16 ENDIF;
17 MassIn := ^/FLOW_IN' * ^/PERIOD_EXEC;
18 FracChng := MassIn / ( MassIn + ^/LIQ_MASS' );
19 ^/LIQ_DENS' := ( ^/DENS_IN' * FracChng ) + ( ^/LIQ_DENS' * (1 - FracChng) );
20 ^/LIQ_TEMP' := ( ^/TEMP_IN' * FracChng ) + ( ^/LIQ_TEMP' * (1 - FracChng) );
21 ^/LIQ_SPHT' := ( ^/SPHT_IN' * FracChng ) + ( ^/LIQ_SPHT' * (1 - FracChng) );
22 ^/LIQ_MASS' := ^/LIQ_MASS' + ( ^/FLOW_IN' - ^/FLOW_OUT' ) * ^/PERIOD_EXEC;
23 ^/LIQ_LEVEL' := ( ^/LIQ_MASS' / ^/LIQ_DENS' ) / ^/TNK_CONS'; (*CUliter/meter/culiter *)
24 IF ( ^/LIQ_LEVEL' > ^/MAX_LEVEL' ) THEN
25   ^/LIQ_LEVEL' := ^/MAX_LEVEL';
26   ^/LIQ_MASS' := ( ^/MAX_LEVEL' * ^/TNK_CONS' ) * ^/LIQ_DENS';
27 ENDIF;
28 IF ( ^/LIQ_LEVEL' > ^/POS_IN' ) THEN
29   ^/BKCAL_OUT_PRES' := ( ^/LIQ_LEVEL' - ^/POS_IN' ) * ^/LIQ_DENS';
30 ELSE
31   ^/BKCAL_OUT_PRES' := 0.0; (*Retain last Dens, Temp, Spht in line *)
32 ENDIF;
33 IF ( ^/LIQ_LEVEL' > ^/POS_OUT' ) THEN
34   ^/PRES_OUT' := ( ^/LIQ_LEVEL' - ^/POS_OUT' ) * ^/LIQ_DENS';
35   ^/DENS_OUT' := ^/LIQ_DENS';
36   ^/TEMP_OUT' := ^/LIQ_TEMP';
37   ^/SPHT_OUT' := ^/LIQ_SPHT';
38 ELSE
39   ^/PRES_OUT' := 0.0;
40 ENDIF;

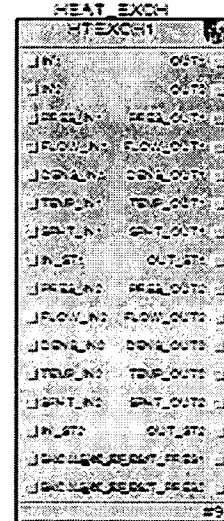
```

4.11 Heat Exchange Element

The heat exchange element is defined as a composite template. This block utilized the following parameters and algorithm.

4.11.1 Heat Exchanger Parameters

Parameter	Default	Connection type	Parameter type
BKCAL_IN_P...	0	Input	Floating point
BKCAL_IN_P...	0	Input	Floating point
BKCAL_OUT...	0	Output	Floating point
BKCAL_OUT...	0	Output	Floating point
DENS_IN1	0	Input	Floating point
DENS_IN2	0	Input	Floating point
DENS_OUT1	0	Output	Floating point
DENS_OUT2	0	Output	Floating point
EXCHG_AREA	100	Internal read only	Floating point
FLOW_IN1	0	Input	Floating point
FLOW_IN2	0	Input	Floating point
FLOW_OUT1	0	Output	Floating point
FLOW_OUT2	0	Output	Floating point
HT_ABSORBED	0	Internal read only	Floating point
IN1		Input	Floating point array
IN2		Input	Floating point array
IN_ST1	No_Connect...	Input	Named Set
IN_ST2	No_Connect...	Input	Named Set
OUT1		Output	Floating point array
OUT2		Output	Floating point array
OUT_ST1	No_Connect...	Output	Named Set
OUT_ST2	No_Connect...	Output	Named Set
PIPE_CONST1	0	Internal read only	Floating point
PIPE_CONST2	0	Internal read only	Floating point
PRES_IN1	0	Input	Floating point
PRES_IN2	0	Input	Floating point
PRES_OUT1	0	Output	Floating point
PRES_OUT2	0	Output	Floating point
SPHT_IN1	0	Input	Floating point
SPHT_IN2	0	Input	Floating point
SPHT_OUT1	0	Output	Floating point
SPHT_OUT2	0	Output	Floating point
TEMP_IN1	0	Input	Floating point
TEMP_IN2	0	Input	Floating point
TEMP_OUT1	0	Output	Floating point
TEMP_OUT2	0	Output	Floating point
TRANS_COEF	1000	Internal read only	Floating point



4.11.2 Heat Exchange Algorithm – Outlet Temperature and Heat Transfer

```

1 IF ( ^/FLOW_IN1' <= 0.0 ) OR ( ^/FLOW_IN2' <= 0.0 ) THEN
2   ^/TEMP_OUT1' := ^/TEMP_IN1';
3   ^/TEMP_OUT2' := ^/TEMP_IN2';
4   ^/HT_ABSORBED' := 0.0;
5 ELSE
6   N1 := ( ^/TRANS_COEF * ^/EXCHG_AREA' ) / ( ^/FLOW_IN1' * ^/SPHT_IN1' );
7   N2 := ( ^/TRANS_COEF * ^/EXCHG_AREA' ) / ( ^/FLOW_IN2' * ^/SPHT_IN2' );
8   N3 := EXP ( N2 * N1 );
9   ^/TEMP_OUT2' := ( ( N3 - 1 ) * ^/TEMP_IN1' + ( 1 - N1 / N2 ) * ^/TEMP_IN2' ) / ( N3 - N1 / N2 );
10  ^/TEMP_OUT1' := ( N3 * ( 1 - N1 / N2 ) * ^/TEMP_IN1' + ( N1 / N2 ) * ( N3 - 1 ) * ^/TEMP_IN2' ) / ( N3 - N1 / N2 );
11  ^/HT_ABSORBED' := ^/FLOW_IN1' * ^/SPHT_IN1' * ( ^/TEMP_IN1' - ^/TEMP_OUT1' );
12 ENDIF;

```

4.11.3 Heat Exchange Algorithm – Flow and Pressure for Input 1

```

1 IF ( ^/IN_ST1' = 'PROC_ST:No_Connection' ) THEN (* If not connected, then propagate value and status *)
2   ^/OUT_ST1' := 'PROC_ST:Good_P_F';
3   ^/PRES_OUT1' := ^/PRES_IN1';
4   ^/FLOW_OUT1' := ^/FLOW_IN1';
5   ^/BKCAL_OUT_PRES1' := ^/BKCAL_IN_PRES1';
6 ELSE
7   IF ( ^/IN_ST1' = 'PROC_ST:Good_F' ) THEN (* Constant flow, propagate pressure flow status *)
8     ^/OUT_ST1' := ^/IN_ST1';
9     ^/PRES_OUT1' := ^/PRES_IN1';
10    ^/FLOW_OUT1' := ^/FLOW_IN1';
11    ^/BKCAL_OUT_PRES1' := ^/BKCAL_IN_PRES1';
12  ELSE
13    IF ( ^/IN_ST1' = 'PROC_ST:Good_P' ) THEN
14      ^/FLOW_OUT1' := 0.0; (* No flow upstream, propagate press *)
15      ^/PRES_OUT1' := ^/PRES_IN1';
16      ^/BKCAL_OUT_PRES1' := ^/BKCAL_IN_PRES1';
17      ^/OUT_ST1' := ^/IN_ST1';
18    ELSE
19      ^/FLOW_OUT1' := ^/FLOW_IN1'; (* Calculate pipe outlet pressure *)
20      ^/PRES_OUT1' := ^/PRES_IN1' * ( ^/FLOW_IN1' * ^/FLOW_IN1' ) * ^/PIPE_CONST1';
21      ^/BKCAL_OUT_PRES1' := ^/PRES_IN1';
22      ^/OUT_ST1' := ^/IN_ST1';
23    ENDIF;
24  ENDIF;
25 ENDIF;
26 ^/DENS_OUT1' := ^/DENS_IN1';
27 ^/SPHT_OUT1' := ^/SPHT_IN1';

```

4.11.4 Heat Exchange Algorithm – Flow and Pressure for Input 2

```

1 IF ( ^/IN_ST2 = 'PROC_ST:No_Connection' ) THEN (* If not connected, then propagate value and status *)
2   ^/OUT_ST2 := 'PROC_ST:Good_P_F';
3   ^/PRES_OUT2 := ^/PRES_IN2;
4   ^/FLOW_OUT2 := ^/FLOW_IN2;
5   ^/BKCAL_OUT_PRES2 := ^/BKCAL_IN_PRES2;
6 ELSE
7   IF ( ^/IN_ST1 = 'PROC_ST:Good_F' ) THEN (* Constant flow, propagate pressure flow status *)
8     ^/OUT_ST2 := ^/IN_ST2;
9     ^/PRES_OUT2 := ^/PRES_IN2;
10    ^/FLOW_OUT2 := ^/FLOW_IN2;
11    ^/BKCAL_OUT_PRES2 := ^/BKCAL_IN_PRES2;
12  ELSE
13    IF ( ^/IN_ST2 = 'PROC_ST:Good_P' ) THEN
14      ^/FLOW_OUT2 := 0.0; (* No flow upstream, propagate press *)
15      ^/PRES_OUT2 := ^/PRES_IN2;
16      ^/BKCAL_OUT_PRES2 := ^/BKCAL_IN_PRES2;
17      ^/OUT_ST2 := ^/IN_ST2;
18    ELSE
19      ^/FLOW_OUT2 := ^/FLOW_IN2; (* Calculate pipe outlet pressure *)
20      ^/PRES_OUT2 := ^/PRES_IN2 - ( ^/FLOW_IN2 * ^/FLOW_IN2 ) * ^/PIPE_CONST2;
21      ^/BKCAL_OUT_PRES2 := ^/PRES_IN2;
22      ^/OUT_ST2 := ^/IN_ST2;
23    ENDIF;
24  ENDIF;
25 ENDIF;
26 ^/DENS_OUT2 := ^/DENS_IN2;
27 ^/SPHT_OUT2 := ^/SPHT_IN2;

```

4.12 Variable Speed Pump Element

The variable speed element is defined as a composite template. This block utilized the following parameters and algorithm.

4.12.1 Variable Speed Pump Parameters

Parameter	Default	Connection type	Parameter type
BKCAL_IN_P...	0	Input	Floating point
BKCAL_OUT...	0	Output	Floating point
DENS_IN	1000	Input	Floating point
DENS_OUT	0	Output	Floating point
FLOW_IN	0	Input	Floating point
FLOW_MAX	31.48	Internal read only	Floating point
FLOW_OUT	0	Output	Floating point
IN		Input	Floating point array
IN_ST	No_Connect...	Input	Named Set
OUT		Output	Floating point array
OUT_ST	No_Connect...	Output	Named Set
PRES_IN	0	Input	Floating point
PRES_OUT	0	Output	Floating point
PUMP_CONST	60	Internal read only	Floating point
PUMP_STATE	On	Input	Named Set
SPEED_PCT	0	Input	Floating point with status
SPHT_IN	4.184	Input	Floating point
SPHT_OUT	0	Output	Floating point
TEMP_IN	30	Input	Floating point
TEMP_OUT	0	Output	Floating point

4.12.2 Variable Speed Pump Algorithm

```

1 IF (^/IN_ST' = 'PROC_ST:No_Connection') THEN (* If not connected, then propagate value and status*)
2   ^/OUT_ST' := 'PROC_ST:Good_P_F';
3   ^/PRES_OUT' := ^/PRES_IN';
4   ^/FLOW_OUT' := ^/FLOW_IN';
5   ^/BKCAL_OUT_PRES' := ^/BKCAL_IN_PRES';
6 ELSE
7   IF (^/IN_ST' = 'PROC_ST:Good_F') THEN (* Constant flow, propagate pressure flow status *)
8     ^/OUT_ST' := ^/IN_ST';
9     ^/PRES_OUT' := ^/PRES_IN';
10    ^/FLOW_OUT' := ^/FLOW_IN';
11    ^/BKCAL_OUT_PRES' := ^/BKCAL_IN_PRES';
12  ELSE
13    IF (^/PUMP_STATE' = 'MOTOR:Off') THEN (* Determine if pump running*)
14      ^/FLOW_OUT' := 0.0; (* Motor off no flow possible *)
15      ^/PRES_OUT' := ^/BKCAL_IN_PRES';
16      ^/BKCAL_OUT_PRES' := ^/PRES_IN';
17      ^/OUT_ST' := 'PROC_ST:Good_P';
18    ELSE
19      IF (^/IN_ST' = 'PROC_ST:Good_P') THEN (* No flow upstream, propagate press *)
20        ^/FLOW_OUT' := 0.0;
21        ^/PRES_OUT' := ^/PRES_IN';
22        ^/BKCAL_OUT_PRES' := ^/PRES_IN';
23        ^/OUT_ST' := ^/IN_ST';
24      ELSE
25        ^/FLOW_OUT' := ^/FLOW_IN';
26        FractionOpen := (^/SPEED_PCT' / 100.0);
27        FIFactor := ((FractionOpen ^ ^/FLOW_MAX') - ^/FLOW_IN'); (* Calculate discharge factor *)
28        IF (FIFactor < 0.0) THEN
29          FIFactor := 0.0;
30        ENDIF;
31        ^/PRES_OUT' := ^/PRES_IN' + (^/PUMP_CONST') * SQRT(FIFactor);
32        ^/BKCAL_OUT_PRES' := ^/PRES_IN';
33        ^/OUT_ST' := ^/IN_ST';
34      ENDIF;
35    ENDIF;
36  ENDIF;
37 ENDIF;
38 ^/DENS_OUT' := ^/DENS_IN';
39 ^/TEMP_OUT' := ^/TEMP_IN';
40 ^/SPHT_OUT' := ^/SPHT_IN';

```

5 Appendix – Conversion Table

The following conversion factors are taken from table B.9 from the NIST publication SP811, Guide to SI Units. These factors may be used in the process block algorithm to convert between US and Metric units.

5.1 Area

To convert from	to	Multiply by	
acre (based on U.S. survey foot)	square meter (m ²)	4.046 873	E+03
<i>hectare</i> (ha)	square meter (m ²)	1.0	E+04
square foot (ft ²)	square meter (m ²)	9.290 304	E-02
square inch (in ²)	square meter (m ²)	6.4516	E-04
square inch (in ²)	square centimeter (cm ²)	6.4516	E+00
square mile (mi ²)	square meter (m ²)	2.589 988	E+06
square mile (mi ²)	square kilometer (km ²)	2.589 988	E+00
square mile (based on U.S. survey foot) (mi ²)	square meter (m ²)	2.589 998	E+06
square mile (based on U.S. survey foot) (mi ²)	square kilometer (km ²)	2.589 998	E+00
square yard (yd ²)	square meter (m ²)	8.361 274	E-01

5.2 Energy (includes Work)

To convert from	to	Multiply by	
British thermal unit _{IT} (Btu _{IT})	joule (J)	1.055 056	E+03
British thermal unit _{th} (Btu _{th})	joule (J)	1.054 350	E+03
British thermal unit (mean) (Btu)	joule (J)	1.055 87	E+03
British thermal unit (39 °F) (Btu)	joule (J)	1.059 67	E+03
British thermal unit (59 °F) (Btu)	joule (J)	1.054 80	E+03
British thermal unit (60 °F) (Btu)	joule (J)	1.054 68	E+03
calorie _{IT} (cal _{IT})	joule (J)	4.1868	E+00
calorie _{th} (cal _{th})	joule (J)	4.184	E+00
calorie (mean) (cal)	joule (J)	4.190 02	E+00
calorie (15 °C) (cal ₁₅)	joule (J)	4.185 80	E+00
calorie (20 °C) (cal ₂₀)	joule (J)	4.181 90	E+00
calorie _{IT} , kilogram (nutrition) <u>12</u>	joule (J)	4.1868	E+03
calorie _{th} , kilogram (nutrition) <u>12</u>	joule (J)	4.184	E+03
calorie (mean), kilogram (nutrition) <u>12</u>	joule (J)	4.190 02	E+03
foot poundal	joule (J)	4.214 011	E-02
foot pound-force (ft · lbf)	joule (J)	1.355 818	E+00
kilocalorie _{IT} (kcal _{IT})	joule (J)	4.1868	E+03

kilocalorie _{th} (kcal _{th})	joule (J)	4.184	E+03
kilocalorie (mean) (kcal)	joule (J)	4.190 02	E+03
kilowatt hour (kW · h)	joule (J)	3.6	E+06
kilowatt hour (kW · h)	megajoule (MJ)	3.6	E+00
watt hour (W · h)	joule (J)	3.6	E+03
watt second (W · s)	joule (J)	1.0	E+00

5.3 Force

To convert from	to	Multiply by	
kilogram-force (kgf)	newton (N)	9.806 65	E+00
kilopond (kilogram-force) (kp)	newton (N)	9.806 65	E+00
kip (1 kip= 1000 lbf)	newton (N)	4.448 222	E+03
kip (1 kip= 1000 lbf)	kilonewton (kN)	4.448 222	E+00
ounce (avoirdupois)-force (ozf)	newton (N)	2.780 139	E-01
poundal	newton (N)	1.382 550	E-01
pound-force (lbf) <u>24</u>	newton (N)	4.448 222	E+00
pound-force per pound (lbf/lb) (thrust to mass ratio)	newton per kilogram (N/kg)	9.806 65	E+00
ton-force (2000 lbf)	newton (N)	8.896 443	E+03
ton-force (2000 lbf)	kilonewton (kN)	8.896 443	E+00

5.4 Heat (Available Energy)

To convert from	to	Multiply by	
British thermal unit _{IT} per cubic foot (Btu _{IT} /ft ³)	joule per cubic meter (J/m ³)	3.725 895	E+04
British thermal unit _{th} per cubic foot (Btu _{th} /ft ³)	joule per cubic meter (J/m ³)	3.723 403	E+04
British thermal unit _{IT} per pound (Btu _{IT} /lb)	joule per kilogram (J/kg)	2.326	E+03
British thermal unit _{th} per pound (Btu _{th} /lb)	joule per kilogram (J/kg)	2.324 444	E+03
calorie _{IT} per gram (cal _{IT} /g)	joule per kilogram (J/kg)	4.1868	E+03
calorie _{th} per gram (cal _{th} /g)	joule per kilogram (J/kg)	4.184	E+03

5.5 Coefficient of Heat Transfer

To convert from	to	Multiply by	
British thermal unit _{IT} per hour square foot degree Fahrenheit [Btu _{IT} /(h · ft ² · °F)]	watt per square meter kelvin [W/(m ² · K)]	5.678 263	E+00
British thermal unit _{th} per hour square foot degree Fahrenheit [Btu _{th} /(h · ft ² · °F)]	watt per square meter kelvin [W/(m ² · K)]	5.674 466	E+00
British thermal unit _{IT} per second square foot degree Fahrenheit [Btu _{IT} /(s · ft ² · °F)]	watt per square meter kelvin [W/(m ² · K)]	2.044 175	E+04
British thermal unit _{th} per second square foot degree Fahrenheit [Btu _{th} /(s · ft ² · °F)]	watt per square meter kelvin [W/(m ² · K)]	2.042 808	E+04

5.6 Density of Heat Flow Rate

To convert from	to	Multiply by
British thermal unit _{IT} per square foot hour [Btu _{IT} /(ft ² · h)]	watt per square meter (W/m ²)	3.154 591 E+00
British thermal unit _{th} per square foot hour [Btu _{th} /(ft ² · h)]	watt per square meter (W/m ²)	3.152 481 E+00
British thermal unit _{th} per square foot minute [Btu _{th} /(ft ² · min)]	watt per square meter (W/m ²)	1.891 489 E+02
British thermal unit _{IT} per square foot second [Btu _{IT} /(ft ² · s)]	watt per square meter (W/m ²)	1.135 653 E+04
British thermal unit _{th} per square foot second [Btu _{th} /(ft ² · s)]	watt per square meter (W/m ²)	1.134 893 E+04
British thermal unit _{th} per square inch second [Btu _{th} /(in ² · s)]	watt per square meter (W/m ²)	1.634 246 E+06
calorie _{th} per square centimeter minute [cal _{th} /(cm ² · min)]	watt per square meter (W/m ²)	6.973 333 E+02
calorie _{th} per square centimeter second [cal _{th} /(cm ² · s)]	watt per square meter (W/m ²)	4.184 E+04

5.7 Heat Capacity and Entropy

To convert from	to	Multiply by
British thermal unit _{IT} per degree Fahrenheit (Btu _{IT} /°F)	joule per kelvin (J/k)	1.899 101 E+03
British thermal unit _{th} per degree Fahrenheit (Btu _{th} /°F)	joule per kelvin (J/k)	1.897 830 E+03
British thermal unit _{IT} per degree Rankine (Btu _{IT} /°R)	joule per kelvin (J/k)	1.899 101 E+03
British thermal unit _{th} per degree Rankine (Btu _{th} /°R)	joule per kelvin (J/k)	1.897 830 E+03

5.8 Heat Flow Rate

To convert from	to	Multiply by
British thermal unit _{IT} per hour (Btu _{IT} /h)	watt (W)	2.930 711 E-01
British thermal unit _{th} per hour (Btu _{th} /h)	watt (W)	2.928 751 E-01
British thermal unit _{th} per minute (Btu _{th} /min)	watt (W)	1.757 250 E+01
British thermal unit _{IT} per second (Btu _{IT} /s)	watt (W)	1.055 056 E+03
British thermal unit _{th} per second (Btu _{th} /s)	watt (W)	1.054 350 E+03
calorie _{th} per minute (cal _{th} /min)	watt (W)	6.973 333 E-02
calorie _{th} per second (cal _{th} /s)	watt (W)	4.184 E+00
kilocalorie _{th} per minute (kcal _{th} /min)	watt (W)	6.973 333 E+01
kilocalorie _{th} per second (kcal _{th} /s)	watt (W)	4.184 E+03
ton of refrigeration (12 000 Btu _{IT} /h)	watt (W)	3.516 853 E+03

5.9 Specific Heat Capacity and Specific Entropy

To convert from	to	Multiply by	
British thermal unit _{IT} per pound degree Fahrenheit [Btu _{IT} /(lb · °F)]	joule per kilogram kelvin [J/(kg · K)]	4.1868	E+03
British thermal unit _{th} per pound degree Fahrenheit [Btu _{th} /(lb · °F)]	joule per kilogram kelvin [J/(kg · K)]	4.184	E+03
calorie _{IT} per gram degree Celsius [cal _{IT} /(g · °C)]	joule per kilogram kelvin [J/(kg · K)]	4.1868	E+03
calorie _{th} per gram degree Celsius [cal _{th} /(g · °C)]	joule per kilogram kelvin [J/(kg · K)]	4.184	E+03

5.10 Thermal Conductivity

To convert from	to	Multiply by	
British thermal unit _{IT} foot per hour square foot degree Fahrenheit [Btu _{IT} · ft/(h · ft ² · °F)]	watt per meter kelvin [W/(m · K)]	1.730 735	E+00
British thermal unit _{th} foot per hour square foot degree Fahrenheit [Btu _{th} · ft/(h · ft ² · °F)]	watt per meter kelvin [W/(m · K)]	1.729 577	E+00
British thermal unit _{IT} inch per hour square foot degree Fahrenheit [Btu _{IT} · in/(h · ft ² · °F)]	watt per meter kelvin [W/(m · K)]	1.442 279	E-01
British thermal unit _{th} inch per hour square foot degree Fahrenheit [Btu _{th} · in/(h · ft ² · °F)]	watt per meter kelvin [W/(m · K)]	1.441 314	E-01
British thermal unit _{IT} inch per second square foot degree Fahrenheit [Btu _{IT} · in/(s · ft ² · °F)]	watt per meter kelvin [W/(m · K)]	5.192 204	E+02
British thermal unit _{th} inch per second square foot degree Fahrenheit [Btu _{th} · in/(s · ft ² · °F)]	watt per meter kelvin [W/(m · K)]	5.188 732	E+02
calorie _{th} per centimeter second degree Celsius [cal _{th} /(cm · s · °C)]	watt per meter kelvin [W/(m · K)]	4.184	E+02

5.11 Thermal Insulance

To convert from	to	Multiply by	
degree Fahrenheit hour square foot per British thermal unit _{IT} (°F · h · ft ² /Btu _{IT})	square meter kelvin per watt (m ² · K/W)	1.761 102	E-01
degree Fahrenheit hour square foot per British thermal unit _{th} (°F · h · ft ² /Btu _{th})	square meter kelvin per watt (m ² · K/W)	1.762 280	E-01

5.12 Thermal Resistance

Thermal Resistance

To convert from	to	Multiply by	
degree Fahrenheit hour per British thermal unit _{IT} (°F · h/Btu _{IT})	kelvin per watt (K/W)	1.895 634	E+00
degree Fahrenheit hour per British thermal unit _{th} (°F · h/Btu _{th})	kelvin per watt (K/W)	1.896 903	E+00

degree Fahrenheit second per British thermal unit_{IT} (°F · s/Btu_{IT}) kelvinper watt (K/W) 5.265 651 E-04
degree Fahrenheit second per British thermal unit_{th} (°F · s/Btu_{th}) kelvinper watt (K/W) 5.269 175 E-04

5.13 Thermal Resistivity

To convert from	to	Multiply by	
degree Fahrenheit hour square foot per British thermal unit _{IT} inch [°F · h · ft ² /(Btu _{IT} · in)]	meter kelvin per watt (m · K/W)	6.933 472	E+00
degree Fahrenheit hour square foot per British thermal unit _{th} inch [°F · h · ft ² /(Btu _{th} · in)]	meter kelvin per watt (m · K/W)	6.938 112	E+04

5.14 Length

To convert from	to	Multiply by	
foot (ft)	meter (m)	3.048	E-01
foot (U.S. survey) (ft)	meter (m)	3.048 006	E-01
inch (in)	meter (m)	2.54	E-02
inch (in)	centimeter (cm)	2.54	E+00
mil (0.001 in)	meter (m)	2.54	E-05
mil (0.001 in)	millimeter (mm)	2.54	E-02
mile (mi)	meter (m)	1.609 344	E+03
mile (mi)	kilometer (km)	1.609 344	E+00
yard (yd)	meter (m)	9.144	E-01

5.15 Mass

To convert from	to	Multiply by	
hundredweight (long, 112 lb)	kilogram (kg)	5.080 235	E+01
hundredweight (short, 100 lb)	kilogram (kg)	4.535 924	E+01
kilogram-force second squared per meter (kgf · s ² /m)	kilogram (kg)	9.806 65	E+00
pound (avoirdupois) (lb) <u>23</u>	kilogram (kg)	4.535 924	E-01
pound (troy or apothecary) (lb)	kilogram (kg)	3.732 417	E-01
pound foot squared (lb · ft ²)	kilogram meter squared (kg · m ²)	4.214 011	E-02
pound inch squared (lb · in ²)	kilogram meter squared (kg · m ²)	2.926 397	E-04
ton, assay (AT)	kilogram (kg)	2.916 667	E-02
ton, assay (AT)	gram (g)	2.916 667	E+01
ton, long (2240 lb)	kilogram (kg)	1.016 047	E+03
ton, metric (t)	kilogram (kg)	1.0	E+03

tonne (called "metric ton" in U.S.) (t)	kilogram (kg)	1.0	E+03
ton, short (2000 lb)	kilogram (kg)	9.071 847	E+02

5.16 Mass Divided by Time (includes Flow)

To convert from	to	Multiply by
pound per hour (lb/h)	kilogram per second (kg/s)	1.259 979 E-04
pound per minute (lb/min)	kilogram per second (kg/s)	7.559 873 E-03
pound per second (lb/s)	kilogram per second (kg/s)	4.535 924 E-01
ton, short, per hour	kilogram per second (kg/s)	2.519 958 E-01

5.17 Mass Divided by Volume (includes Mass Density and Concentration)

To convert from	to	Multiply by
pound per cubic foot (lb/ft ³)	kilogram per cubic meter (kg/m ³)	1.601 846 E+01
pound per cubic inch (lb/in ³)	kilogram per cubic meter (kg/m ³)	2.767 990 E+04
pound per cubic yard (lb/yd ³)	kilogram per cubic meter (kg/m ³)	5.932 764 E-01
pound per gallon [Canadian and U.K. (Imperial)] (lb/gal)	kilogram per cubic meter (kg/m ³)	9.977 637 E+01
pound per gallon [Canadian and U.K. (Imperial)] (lb/gal)	kilogram per liter (kg/L)	9.977 637 E-02
pound per gallon (U.S.) (lb/gal)	kilogram per cubic meter (kg/m ³)	1.198 264 E+02
pound per gallon (U.S.) (lb/gal)	kilogram per liter (kg/L)	1.198 264 E-01
slug per cubic foot (slug/ft ³)	kilogram per cubic meter (kg/m ³)	5.153 788 E+02
ton, long, per cubic yard	kilogram per cubic meter (kg/m ³)	1.328 939 E+03
ton, short, per cubic yard	kilogram per cubic meter (kg/m ³)	1.186 553 E+03

5.18 Power

To convert from	to	Multiply by
foot pound-force per hour (ft · lbf/h)	watt (W)	3.766 161 E-04
foot pound-force per minute (ft · lbf/min)	watt (W)	2.259 697 E-02
foot pound-force per second (ft · lbf/s)	watt (W)	1.355 818 E+00
horsepower (550 ft · lbf/s)	watt (W)	7.456 999 E+02
horsepower (boiler)	watt (W)	9.809 50 E+03
horsepower (electric)	watt (W)	7.46 E+02
horsepower (metric)	watt (W)	7.354 988 E+02
horsepower (U.K.)	watt (W)	7.4570 E+02
horsepower (water)	watt (W)	7.460 43 E+02

5.19 Pressure or Stress (Force divided by area)

To convert from	to	Multiply by	
atmosphere, standard (atm)	pascal (Pa)	1.013 25	E+05
atmosphere, standard (atm)	kilopascal (kPa)	1.013 25	E+02
atmosphere, technical (at)	pascal (Pa)	9.806 65	E+04
atmosphere, technical (at)	kilopascal (kPa)	9.806 65	E+01
bar (bar)	pascal (Pa)	1.0	E+05
bar (bar)	kilopascal (kPa)	1.0	E+02
centimeter of mercury (0 °C)	pascal (Pa)	1.333 22	E+03
centimeter of mercury (0 °C)	kilopascal (kPa)	1.333 22	E+00
centimeter of mercury, conventional (cmHg)	pascal (Pa)	1.333 224	E+03
centimeter of mercury, conventional (cmHg)	kilopascal (kPa)	1.333 224	E+00
centimeter of water (4 °C)	pascal (Pa)	9.806 38	E+01
centimeter of water, conventional (cmH ₂ O)	pascal (Pa)	9.806 65	E+01
dyne per square centimeter (dyn/cm ²)	pascal (Pa)	1.0	E-01
foot of mercury, conventional (ftHg)	pascal (Pa)	4.063 666	E+04
foot of mercury, conventional (ftHg)	kilopascal (kPa)	4.063 666	E+01
foot of water (39.2 °F)	pascal (Pa)	2.988 98	E+03
foot of water (39.2 °F)	kilopascal (kPa)	2.988 98	E+00
foot of water, conventional (ftH ₂ O)	pascal (Pa)	2.989 067	E+03
foot of water, conventional (ftH ₂ O)	kilopascal (kPa)	2.989 067	E+00
gram-force per square centimeter (gf/cm ²)	pascal (Pa)	9.806 65	E+01
inch of mercury (32 °F)	pascal (Pa)	3.386 38	E+03
inch of mercury (32 °F)	kilopascal (kPa)	3.386 38	E+00
inch of mercury (60 °F)	pascal (Pa)	3.376 85	E+03
inch of mercury (60 °F)	kilopascal (kPa)	3.376 85	E+00
inch of mercury, conventional (inHg)	pascal (Pa)	3.386 389	E+03
inch of mercury, conventional (inHg)	kilopascal (kPa)	3.386 389	E+00
inch of water (39.2 °F)	pascal (Pa)	2.490 82	E+02
inch of water (60 °F)	pascal (Pa)	2.4884	E+02
inch of water, conventional (inH ₂ O)	pascal (Pa)	2.490 889	E+02
kilogram-force per square centimeter (kgf/cm ²)	pascal (Pa)	9.806 65	E+04
kilogram-force per square centimeter (kgf/cm ²)	kilopascal (kPa)	9.806 65	E+01
kilogram-force per square meter (kgf/m ²)	pascal (Pa)	9.806 65	E+00
kilogram-force per square millimeter (kgf/mm ²)	pascal (Pa)	9.806 65	E+06
kilogram-force per square millimeter (kgf/mm ²)	megapascal (MPa)	9.806 65	E+00
kip per square inch (ksi) (kip/in ²)	pascal (Pa)	6.894 757	E+06
kip per square inch (ksi) (kip/in ²)	kilopascal (kPa)	6.894 757	E+03
millibar (mbar)	pascal (Pa)	1.0	E+02
millibar (mbar)	kilopascal (kPa)	1.0	E-01
millimeter of mercury, conventional (mmHg) <u>13</u>	pascal (Pa)	1.333 224	E+02
millimeter of water, conventional (mmH ₂ O) <u>13</u>	pascal (Pa)	9.806 65	E+00

poundal per square foot	pascal (Pa)	1.488 164 E+00
pound-force per square foot (lbf/ft ²)	pascal (Pa)	4.788 026 E+01
pound-force per square inch (psi) (lbf/in ²)	pascal (Pa)	6.894 757 E+03
pound-force per square inch (psi) (lbf/in ²)	kilopascal (kPa)	6.894 757 E+00
psi (pound-force per square inch) (lbf/in ²)	pascal (Pa)	6.894 757 E+03
psi (pound-force per square inch) (lbf/in ²)	kilopascal (kPa)	6.894 757 E+00
torr (Torr)	pascal (Pa)	1.333 224 E+02

5.20 Temperature

To convert from	to	Multiply by
<i>degree Celsius</i> (°C)	kelvin (K)	$T/K = t/^{\circ}\text{C} + 273.15$
degree centigrade <u>16</u>	degree Celsius (°C)	$t/^{\circ}\text{C} \approx t/\text{deg. cent.}$
degree Fahrenheit (°F)	degree Celsius (°C)	$t/^{\circ}\text{C} = (t/^{\circ}\text{F} - 32)/1.8$
degree Fahrenheit (°F)	kelvin (K)	$T/K = (t/^{\circ}\text{F} + 459.67)/1.8$
degree Rankine (°R)	kelvin (K)	$T/K = (T/^{\circ}\text{R})/1.8$
<i>kelvin</i> (K)	degree Celsius (°C)	$t/^{\circ}\text{C} = T/K - 273.15$

5.21 Temperature Interval

To convert from	to	Multiply by
<i>degree Celsius</i> (°C)	kelvin (K)	1.0 E+00
degree centigrade <u>16</u>	degree Celsius (°C)	1.0 E+00
degree Fahrenheit (°F)	degree Celsius (°C)	5.555 556 E-01
degree Fahrenheit (°F)	kelvin (K)	5.555 556 E-01
degree Rankine (°R)	kelvin (K)	5.555 556 E-01

5.22 Time

To convert from	to	Multiply by
<i>day</i> (d)	second (s)	8.64 E+04
day (sidereal)	second (s)	8.616 409 E+04
<i>hour</i> (h)	second (s)	3.6 E+03
hour (sidereal)	second (s)	3.590 170 E+03
<i>minute</i> (min)	second (s)	6.0 E+01
minute (sidereal)	second (s)	5.983 617 E+01
second (sidereal)	second (s)	9.972 696 E-01
shake	second (s)	1.0 E-08
shake	nanosecond (ns)	1.0 E+01
year (365 days)	second (s)	3.1536 E+07
year (sidereal)	second (s)	3.155 815 E+07

year (tropical) second (s) 3.155 693 E+07

5.23 Velocity (includes Speed)

To convert from	to	Multiply by	
foot per hour (ft/h)	meter per second (m/s)	8.466 667	E-05
foot per minute (ft/min)	meter per second (m/s)	5.08	E-03
foot per second (ft/s)	meter per second (m/s)	3.048	E-01
inch per second (in/s)	meter per second (m/s)	2.54	E-02
<i>kilometer per hour</i> (km/h)	meter per second (m/s)	2.777 778	E-01
<i>knot</i> (nautical mile per hour)	meter per second (m/s)	5.144 444	E-01
mile per hour (mi/h)	meter per second (m/s)	4.4704	E-01
mile per hour (mi/h)	kilometer per hour (km/h)	1.609 344	E+00
mile per minute (mi/min)	meter per second (m/s)	2.682 24	E+01
mile per second (mi/s)	meter per second (m/s)	1.609 344	E+03
revolution per minute (rpm) (r/min)	radian per second (rad/s)	1.047 198	E-01
rpm (revolution per minute) (r/min)	radian per second (rad/s)	1.047 198	E-01

5.24 Viscosity, Dynamic

To convert from	to	Multiply by	
centipoise (cP)	pascal second (Pa · s)	1.0	E-03
poise (P)	pascal second (Pa · s)	1.0	E-01
poundal second per square foot	pascal second (Pa · s)	1.488 164	E+00
pound-force second per square foot (lbf · s/ft ²)	pascal second (Pa · s)	4.788 026	E+01
pound-force second per square inch (lbf · s/in ²)	pascal second (Pa · s)	6.894 757	E+03
pound per foot hour [lb/(ft · h)]	pascal second (Pa · s)	4.133 789	E-04
pound per foot second [lb/(ft · s)]	pascal second (Pa · s)	1.488 164	E+00
rhe	reciprocal pascal second [(Pa · s) ⁻¹]	1.0	E+01
slug per foot second [slug/(ft · s)]	pascal second (Pa · s)	4.788 026	E+01

5.25 Viscosity, Kinematic

To convert from	to	Multiply by	
centistokes (cSt)	meter squared per second (m ² /s)	1.0	E-06
square foot per second (ft ² /s)	meter squared per second (m ² /s)	9.290 304	E-02
stokes (St)	meter squared per second (m ² /s)	1.0	E-04

5.26 Volume (includes Capacity)

To convert from	to	Multiply by	
acre-foot (based on U.S. survey foot) <u>2</u>	cubic meter (m ³)	1.233 489	E+03
barrel [for petroleum, 42 gallons (U.S.)](bbl)	cubic meter (m ³)	1.589 873	E-01
barrel [for petroleum, 42 gallons (U.S.)](bbl)	liter (L)	1.589 873	E+02
cubic foot (ft ³)	cubic meter (m ³)	2.831 685	E-02
cubic inch (in ³) <u>14</u>	cubic meter (m ³)	1.638 706	E-05
cubic mile (mi ³)	cubic meter (m ³)	4.168 182	E+09
cubic yard (yd ³)	cubic meter (m ³)	7.645 549	E-01
gallon [Canadian and U.K. (Imperial)] (gal)	cubic meter (m ³)	4.546 09	E-03
gallon [Canadian and U.K. (Imperial)] (gal)	liter (L)	4.546 09	E+00
gallon (U.S.) (gal)	cubic meter (m ³)	3.785 412	E-03
gallon (U.S.) (gal)	liter (L)	3.785 412	E+00
liter (L) <u>20</u>	cubic meter (m ³)	1.0	E-03
ton, register	cubic meter (m ³)	2.831 685	E+00

5.27 Volume Divided by Time (includes Flow)

To convert from	to	Multiply by	
cubic foot per minute (ft ³ /min)	cubic meter per second (m ³ /s)	4.719 474	E-04
cubic foot per minute (ft ³ /min)	liter per second (L/s)	4.719 474	E-01
cubic foot per second (ft ³ /s)	cubic meter per second (m ³ /s)	2.831 685	E-02
cubic inch per minute (in ³ /min)	cubic meter per second (m ³ /s)	2.731 177	E-07
cubic yard per minute (yd ³ /min)	cubic meter per second (m ³ /s)	1.274 258	E-02
gallon (U.S.) per day (gal/d)	cubic meter per second (m ³ /s)	4.381 264	E-08
gallon (U.S.) per day (gal/d)	liter per second (L/s)	4.381 264	E-05
gallon (U.S.) per minute (gpm) (gal/min)	cubic meter per second (m ³ /s)	6.309 020	E-05
gallon (U.S.) per minute (gpm) (gal/min)	liter per second (L/s)	6.309 020	E-02

1. Display

Display – things that you can define as part of the display

Edit Dynamics – any static item can be turned into a dynamic item. When this happens the user then defines a tag and later defines what they want the dynamic behavior to be.

UI Controls

Button, Check-box, Slider, etc

Behaviors

Action, Alert (attach a data mapping)

Conditions

If, Switch

Element

Copy, Create, Move, Replace

Listener (register for an event and take action when the event arrives)

LoadURL

LoadXML

printElement

Attributes

Context Menu, Drag/Drop, Focus, Tool tip, Zoom and Span, Share

Elements

Shape Elements

Rectangle, Circle, Ellipse, Line, Polygon, Polylines, Path

Container Elements

Svg, Group, Symbols

Text Elements

Text, Text path

Reference Elements

Image

Other Elements

Script, PCDATA, description

Cut/Copy/Paste/Delete Nodes

Move to Back/Front

Move One Forward/Backward

2. Groups

Step 1 : Create Inputs and Outputs

Step 2 : Define Functionality for each of the Dynamic Elements (the elements that were given dynamic behavior are defined when the display is built)

Step 3 : Bind Inputs and Outputs to the Functionality associated with each Dynamic Element

3. Data Bindings

Bind inputs and outputs to Data Source

4. Data Binding Sources

The sources of the information that we bind to should be

- 1- DeltaV Runtime
- 2- DeltaV Historian
- 3- OPC
- 4- XML File
- 5- HTTP Browse Link
- 6- Alarm Services
- 7- Event Services
- 8- General Web Services
- 9- Yukon Managed Runtime (ADO interface)